



Facultad de Ciencias

Configuración y monitorización de entorno virtualizado de red

(Configuring and monitoring
virtualized network environment)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Cristian Sainz Diego
Director: Jose Angel Irastorza Teja
Co-Director: Alberto Eloy García Gutiérrez
Septiembre 2021

Agradecimientos

Quisiera agradecer el apoyo ofrecido por mis dos profesores en este proyecto, Jose Angel Irastorza y Alberto Eloy García, por haber estado disponibles en todo momento durante este curso, así como al resto de profesores que han formado parte de esta etapa a lo largo de los años.

Resumen

En los últimos años se está viendo un creciente interés por el cloud computing (para abstraer y centralizar el almacenamiento de información y servicios, siendo accesibles por cualquier usuario con Internet), principalmente debido al exponencial crecimiento de dispositivos, lo cual lleva a adoptar soluciones unificadas y disponibles en todo momento para satisfacer las necesidades imperantes. En este contexto, nacen tecnologías que permiten desplegar un entorno de red altamente configurable y virtualizado que sirva como cluster o centro de datos.

En este proyecto se va a abordar esta problemática, llevando a cabo la descarga y configuración de la herramienta empleada para el despliegue del datacenter. Para ello, se elegirán razonadamente las interfaces de red, creándose y programándose sus puertos de entrada y salida, además de implementando la redirección de datos (flujo de datos) desde el ordenador host hasta la arquitectura virtualizada, siendo para ello necesario abrir los puertos requeridos únicamente y definir las rutas que conforman el flujo. Finalmente, se creará la topología dada, con todo el diseño de seguridad, redes, subredes, routers y disponibilidad de recursos hardware que la propia herramienta permitirá adaptar, tanto por sus ficheros de configuración como por medio de comandos a la hora del despliegue.

Una vez hecho esto se monitorizará la topología creada, por medio de otra herramienta de gestión de red, con la intención de poder recoger el estado de la infraestructura (tanto en general como en sus diferentes componentes) en tiempo real que, en efecto, era el objetivo inicial del experimento para este proyecto.

Palabras clave:

Computación, Red, OpenStack, Nube, Monitorización.

Abstract

In recent years there is a growing interest in cloud computing (to abstract and centralize the storage of information and services, being accessible by any user with the Internet), mainly due to the exponential growth of devices, which leads to the adoption of unified and available solutions at all times to meet the prevailing needs. In this context, technologies are born that allow deploying a highly configurable and virtualized network environment that serves as a cluster or data center.

This project will address this problem, carrying out the download and configuration of the tool used for the deployment of the datacenter. To do this, the network interfaces will be chosen reasonably, creating and programming their input and output ports, in addition to implementing data redirection (data flow) from the host computer to the virtualized architecture, being necessary for this to open the required ports only and define the routes that make up the flow. Finally, the given topology will be created, with all the security design, networks, subnets, routers and availability of hardware resources that the tool itself will allow to adapt, both by its configuration files and by means of commands at the time of deployment.

Once this is done, the topology created will be monitored through another network management tool, with the intention of being able to collect the state of the infrastructure (both in general and in its different components) in real time which, in effect, was the initial objective of the experiment for this project.

Keywords:

Computing, Network, OpenStack, Cloud, Monitoring.

Contents

1	Introducción	5
1.1	Motivación	5
1.2	Objetivos	6
1.3	Estructura del proyecto	6
2	Marco teórico	7
2.1	OpenStack	8
2.2	Nagios	16
3	Marco práctico	18
3.1	OpenStack - Infraestructura	18
3.1.1	Elección de alternativas OpenStack	19
3.1.2	Interfaces de red para el cluster: Motivos y diferencias entre los distintos tipos	19
3.1.3	Configuración inicial	21
3.1.4	Programación de interfaces de red	22
3.1.5	BR-EX: Reenvío de paquetes	24
3.1.6	Redirección de flujo en Windows	29
3.1.7	Definición de topología OpenStack	31
3.1.8	Prueba de funcionamiento con PuTTY	34
3.1.9	Problemas comunes	36
3.2	Nagios - Monitorización del cluster	38
3.2.1	Implementación del core/server	38
3.2.2	Generación de agentes para hosts remotos	41
3.2.3	Demostración de funcionamiento	43
4	Conclusión	46
5	Futuras líneas de trabajo	47
6	Bibliografía	49
7	Anexo	51
7.1	Mano (OSM)	51
7.2	Entorno distribuido (Modularizado)	53
7.3	Entorno con kubernetes/docker	54
7.4	Script de servicios Openstack	55

1 Introducción

Los avances tecnológicos, así como de población, han llevado a un incremento exponencial en los últimos años del hardware total producido y empleado, así como de las redes.

Por ello, el próximo paso a seguir es deshacerse de la carga física en todos estos dispositivos, que cada vez tienen acceso a más servicios e información.

Esto se consigue por medio de la centralización de un entorno de cómputo que almacene todos los servicios e información necesarios para que el usuario final solamente deba conectarse, sin necesidad de tener que guardar dicha información en su propio dispositivo. Como es de esperar, de esta manera se puede ahorrar en mantenimiento de equipo, además de los beneficios de tener accesible en todo momento tu información. Este servicio encargado de centralizar todo es conocido como la Nube (Cloud).

1.1 Motivación

Debido a sus beneficios de cara a las empresas y usuarios, el Cloud ha conseguido asentar su lugar excepcionalmente, en base a tecnologías como Google Drive, enfocadas a nivel de usuario puramente, esto es, a la centralización de datos para persistir sin necesidad del almacenamiento por parte del usuario.

Consecuentemente, aún se puede mejorar la situación, aprovechando la Nube para avanzar y potenciar, esta vez, el dimensionamiento de redes, junto a la integración con las herramientas empleadas en su gestión y monitorización.

Teniendo esto en cuenta, surgen diferentes topologías de red que permitan sacar beneficio de los objetivos de una organización, ya que todas no son iguales ni necesitan los mismos recursos (o estos mismos recursos pero distribuidos igual). Véase, por ejemplo, el caso de una empresa que tenga una aplicación autogestionada. Si la empresa dispone de los recursos, deberá tomar una decisión acerca del uso de los computadores, de modo que una parte de ellos serán para correr módulos, otros para actuar de servidores, otros para filtrar el tráfico, llegando a escalar hasta infinitas posibilidades en función del uso que se le quiera dar así como de los recursos y limitaciones económicas existentes.

En el caso de este proyecto, la principal ambición es llevar a cabo el dimensionamiento y configuración de una red básica, con un servidor y una serie de clientes conectados al mismo, de manera que no va a escalar, ya que se busca que sea un ejemplo práctico y moldeable, con el cual poder entender el proceso de creación desde cero, pasando por el diseño, configuración, integración y monitorización para, finalmente, llegar a una arquitectura coherente y funcional.

1.2 Objetivos

El objetivo final es poder llevar a cabo la monitorización de un entorno de red desplegado propiamente, con la intención de saber en todo momento de qué manera está funcionando, haciendo para ello uso del software Nagios. Consecuentemente, se pueden solucionar los problemas encontrados con la rapidez otorgada por una herramienta de gestión de este tipo.

En cualquier caso, para ello la red Cloud desplegada deberá ser simple así como replicable (portable) en cuanto a su configuración, además de servir como entorno de pruebas o laboratorio, debido mayoritariamente a que no requiere una carga computacional grande (por estar desplegada sobre el propio ordenador nativo). Sin embargo, si bien existe esta limitación de recursos, es obligatorio que la red sea segura y estable, para que el funcionamiento sea óptimo y se pueda trabajar y modificar adecuadamente.

Para llevar a cabo dicho despliegue o infraestructura se pretende emplear como solución el software OpenStack, para funcionar a modo de proveedor Cloud (diseño de topología de red), siendo todas las instancias creadas preferiblemente de Linux (CentOS/Ubuntu) por razones de compatibilidad.

1.3 Estructura del proyecto

Se ha desarrollado el proyecto en 5 capítulos, siendo el primero de todos la Introducción que se acaba de ver, esto es, el capítulo actual.

Posteriormente, el segundo capítulo será el Marco Teórico del proyecto, esto es, una explicación más profunda de las herramientas empleadas para desglosar su funcionamiento y saber enfocar el proyecto en un futuro, lo cual nos lleva al tercer capítulo, que sería el Marco Práctico. En su contenido se van a tratar detalladamente todas las configuraciones, instalaciones, pruebas y problemas encontrados, acompañados de imágenes para hacer más sencillo el entendimiento del documento.

Finalmente, habría un cuarto capítulo para la conclusión, siendo aquí donde se determinan los objetivos conseguidos. A su vez, este capítulo daría paso al último del proyecto, que sería el de Futuras líneas de trabajo, enfocado a describir las posibles soluciones que se podrían implementar en proyectos posteriores, siendo acompañadas por una breve explicación de sus beneficios.

Con estos capítulos se cubre por completo el proyecto, a excepción de la Bibliografía y el Anexo, que no se consideran capítulos en sí.

2 Marco teórico

En este apartado inicial se va a hacer énfasis en las tecnologías que más tarde se utilizarán. Para ello se llevará a cabo un análisis en cierta profundidad de aspectos teóricos que involucran tanto la funcionalidad, módulos y formas de uso, dándole de esta manera una visión general al por qué del uso de cada una.

Sin embargo, primero es necesario presentar adecuadamente las tecnologías. Para ello, en primer lugar recuérdese que este proyecto pretende alcanzar la monitorización de un entorno virtualizado que se creará propiamente, simulando un entorno empresarial cualquiera de la actualidad. Dicho entorno va a pertenecer a la categoría de Cloud Computing, ya que se van a alojar futuramente servicios que la empresa venderá sin que el usuario tenga que encargarse de nada, a diferencia del Cloud normal que lo que hace es guardar datos. Posteriormente, debe aclararse también que inicialmente se va a tratar, a su vez, de un Cloud privado, ya que se va a tratar de dar seguridad al alojamiento de los servicios que se dan y el entorno estaría en el datacenter de la propia empresa. Dicho entorno también podría ser público si fuese accesible para dar uso de ello, pero en este caso no interesa, ya que se trata de un servicio o lógica de mercado interna y es la propia empresa la que debe tratarlo.

Ahora que ya se ha expuesto todo concepto básico de la estructura, se va a pasar a desarrollar la idea de proveedor Cloud, esto es, la manera de poder crear el entorno virtualizado o Cloud. Dicho proveedor es la empresa que se emplea para poder llevar a cabo el servicio que se busca. Existen varios:

- **Amazon Web Services:** Es una plataforma segura que proporciona un amplio conjunto de servicios de infraestructura para almacenamiento, potencia total de cómputo, redes y persistencia en las bases de datos. Una vez se paga, está disponible inmediatamente lo que se ha contratado. [1]
- **Google Cloud Platform:** Entorno que reúne diferentes productos de Google que previamente se ofrecían por separado. Es como un panel de administración con todo lo necesario para correr una empresa. [2]
- **Microsoft Azure:** Es una nube pública de pago en base al uso dado. Permite implementar y administrar aplicaciones en servidores globales de Microsoft a lo largo del globo, esto es, en una red global. [3]
- **Openstack:** Conjunto de herramientas enfocadas a construir y administrar Plataformas de Cómputo en Red para nubes privadas y públicas. Está considerada como el futuro del Cloud. [4]

El proveedor que se ha elegido es Openstack, ya que se trata de Software Libre y sirve sobradamente para llevar a cabo la implementación buscada.

Finalmente, también es crucial entender las herramientas de gestión, que permiten monitorizar el entorno para saber cómo es la experiencia del usuario

y tener un control total de lo que está pasando, de modo que se pueda cambiar de proveedor en el caso de no tenerse un rendimiento adecuado:

- **Zabbix:** Herramienta software que monitoriza diferentes componentes: redes, servidores, máquinas virtuales y servicios Cloud. Presenta métricas como el uso de CPU y disco así como de la propia red y se puede configurar mediante XML para monitorizar tanto Windows como Ubuntu. Está enfocada al soporte y sus plugins de métricas a monitorizar son técnicamente complejos pero gratuitos. [5]
- **Pandora FMS:** Destaca por ser una solución integrada y horizontal de monitorización enfocada a las empresas, lo cual permite poder adaptar diferentes ámbitos de monitorización, referentes a los departamentos, lo cual la convierte en una herramienta muy flexible. [6]
- **Nagios:** Permite descargar plugins ya automatizados, así como poder crear otros, con la intención de monitorizar determinados procesos. A diferencia de Zabbix, una práctica habitual consiste en comercializar plugins prefabricados para medir métricas. [7]

La solución de monitorización escogida es Nagios, por ser Software Libre flexible a la hora de configurar y por permitir medir métricas básicas eficientemente.

2.1 OpenStack

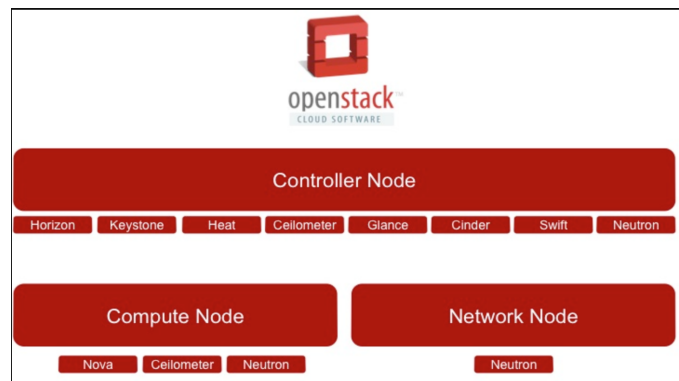


Figure 1: Módulos de OpenStack

OpenStack es un conjunto de soluciones software que permiten diseñar y administrar entornos de computación para redes de todo tipo y alcance. Se suele desplegar como IaaS (Infraestructura como Servicio), que es la clasificación de aquellos servicios con muchas APIs que permiten acceder a detalles de bajo nivel del entorno.

Por otra parte, consigue esto por medio de una serie de posibilidades, como puede ser el caso de desplegar máquinas virtuales o instancias con sus configuraciones correspondientes. Esto hace que el escalado horizontal (muchas máquinas por subred en vez de agrupaciones datacenter-cluster-rack-blade) sea muy sencillo, lo cual permite beneficiarse de tareas que sean concurrentes, esto es, puedan dividirse y ejecutar en varias instancias a la vez.

Finalmente se pasará a explicar los módulos de OpenStack [8] (*Figura 1*), siendo cada uno responsable de una parte de la funcionalidad total, de modo que entre todos se tiene una ejecución completa. Resumidamente serían:

- **Nova:** Gestor principal. Proporciona máquinas virtuales para el entorno.
- **Glance:** Consiste en un repositorio de imágenes de discos virtuales.
- **Neutron:** Proporciona conectividad de red como servicio entre dispositivos que gestionan los servicios de Openstack.
- **Keystone:** Permite autenticar y autorizar los servicios de Openstack.
- **Cinder:** Proporciona almacenamiento de bloques persistente para alojamiento de máquinas virtuales.
- **Swift:** Proporciona almacenamiento escalable y persistencia de objetos.
- **Heat:** Permite emplear servicios de orquestación para múltiples aplicaciones en la nube.

<i>Nova (Compute)</i>

Es el componente gestor por definición, aquel encargado de manejar el resto de módulos, de ahí su complejidad. Se puede desplegar de manera centralizada o distribuida (en diferentes instancias servidoras) y controla los hipervisores para saber desde dónde se lanzarán las instancias. Los requisitos para lanzar una instancia son: tener una imagen de sistema operativo, una red para poder subirla e instalarla en el cluster así como un flavor (sabor) para designar la cantidad de recursos computaciones (hardware).

Los procesos a destacar de este módulo son:

- **nova-api:** Es un handler, al aceptar/rechazar (responder) las llamadas del usuario por parte del uso de la API. Se encarga de, en caso de ser aceptadas, iniciar las máquinas virtuales.
- **nova-volume:** Administra la creación y eliminación de volúmenes asociados a instancias, de modo que se guarde el estado de la máquina en cuanto a almacenamiento se refiere. En versiones actuales el módulo Cinder lleva a que este proceso sea deprecated.
- **nova-compute:** Proceso del propio módulo. Lleva a cabo la creación y eliminación de instancias, así como la ejecución de comandos.

- **nova-network:** Lleva a cabo el control de las peticiones a nivel de red que serán ejecutadas en las instancias o infraestructura.
- **nova-schedule:** Lleva a cabo la planificación, decidiendo en qué instancia se ejecuta una petición aceptada previamente.
- **nova-api-metadata:** Ejecuta peticiones de metadatos.
- **nova-conductor:** Media entre nova-compute y la base de datos.

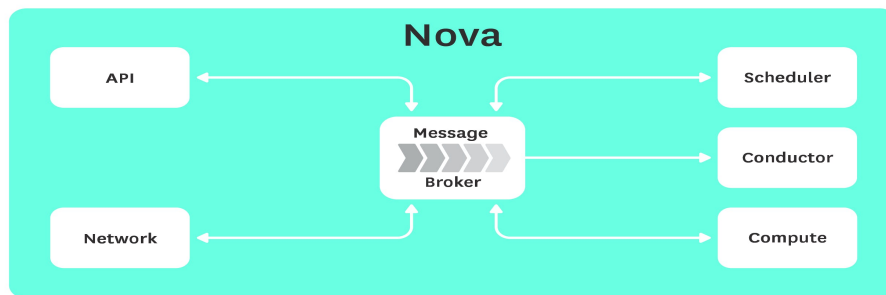


Figure 2: Estructura del módulo Nova

Dichos módulos aparecen en la *Figura 2*, en la cual se puede ver como el *Broker* (la directiva en sí) se relaciona con todos bidireccionalmente, salvo el *conductor*, debido a que este solamente direcciona hacia la base de datos, recibiendo del broker la información del módulo *compute*.

Glance

Componente encargado de llevar a cabo la gestión de las imágenes asociadas a las instancias. Recuérdese que una imagen es un archivo con el código del Sistema Operativo contenido que se instalará en una máquina nueva para correr. Si bien es cierto que vienen algunas predefinidas en OpenStack, normalmente tendrían una carga computacional demasiado baja, con un rendimiento bajo, quedándose limitada para Sistemas Operativos más pesados. Por tanto, es interesante crear imágenes propias para el desarrollo.

Cabe destacar que cuando se está creando una imagen hay varios estados (queued - esperando a ser añadida, saving - los datos están siendo añadidos, active - imagen procesada por completo, deleted - imagen borrada), si bien lo que realmente sería interesante explicar es el formateo de las imágenes a la hora de ser creadas para poder ser utilizadas por OpenStack.

- **vhd:** Formato representativo de la unidad de disco duro virtual.
- **iso:** Imagen con datos en formato ISO. Se puede relacionar con los DVDs.

- **qcow2:** Formato soportado por QEMU que permite expandirse dinámicamente. Este va a ser el formato empleado en la creación que se llevará a cabo durante la configuración, más adelante.
- **vdi:** Formato soportado por QEMU y Virtual Box.

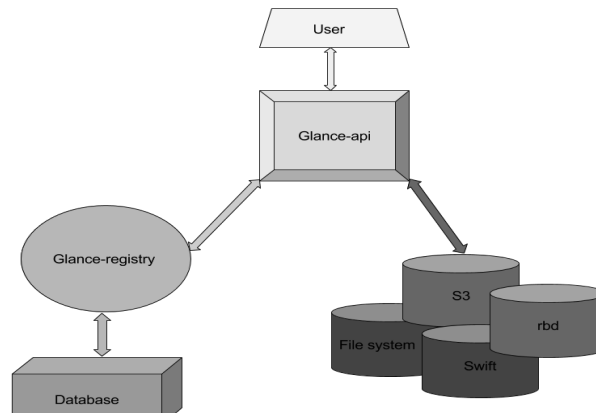


Figure 3: Estructura del módulo Glance

En la *Figura 3* se ve de qué forma un comando por parte del usuario se maneja en la API, la cual se conecta con el registro que guarda las imágenes, así como con otra serie de contenedores como *Swift*.

Neutron (Red)

Se encarga de la gestión de red, llevando a cabo el control de las redes y subredes del entorno virtualizado. Una de las claves de su funcionamiento es la abstracción de la red física o local perteneciente al propio hardware (máquina servidora original, no virtualizada) sobre la que se descarga e implementa OpenStack. Es el módulo que ofrece un servicio NaaS (Networking as a Service).

Entre sus funcionalidades se encuentran la creación de redes con Firewalls, VPNs, balanceo de carga y redes forwarding. Hay dos tipos de redes principales:

- **Provider network:** Redes controladas por el administrador de OpenStack. Es posible que gestionen las redes externas, como ya se verá.
- **Tenant network:** Conocidas como redes de proyectos, son administradas por cualquier miembro del proyecto en cuestión y son inherentes a este. Por tanto, permiten crear redes internas en dicho proyecto, pero sin poder manipular las redes externas mapeadas al hardware original.

Además de haber diferentes tipos de redes, en función de la configuración que se implemente finalmente, los tipos de red concretamente también pueden ser:

- **Flat:** Implementación sencilla pero peligrosa. No es muy segura ya que todas las máquinas virtuales serán visibles dentro del cloud, por lo cual se suelen emplear en redes privadas.
- **LAN:** Red virtual que permite transmitir paquetes con visibilidad de la red interna, de modo que no se pueden enrutar a través de otras redes locales (transmitir de una red a la otra). Se emplean cuando se requieren hacer pruebas de tipo single-node (abstraer la ejecución a un nodo).
- **VLAN:** Normalmente empleada en redes privadas grandes o redes públicas pequeñas. La parte beneficiosa es que las empresas suelen apostar por servicios VLAN, si bien hay que activar el modo trunk en los puertos de cada host físico para permitir funcionar de este modo. A su vez, cada proyecto es asignado a una VLAN.
- **GRE:** Una red virtual empleada para encapsular paquetes de red a modo de túnel. Dichos paquetes son dirigidos por el router en base a la tabla IP del router. Por lo tanto, las redes GRE no se asocian por medio de Networking a redes físicas específicas.
- **VXLAN:** Protocolo de encapsulado para recubrir con una capa más la ya existente capa 3 (Infraestructura TCP/IP). Esto permite habilitar arquitecturas de cómputo elásticas y dinámicas.

Además de poder crear routers, redes y configuraciones muy variadas, este módulo permite crear configuraciones de seguridad, desde grupos con diferentes permisos hasta abrir o cerrar puertos en las diferentes instancias, de modo que se filtren los paquetes en función del servicio y puerto que estén sirviendo.

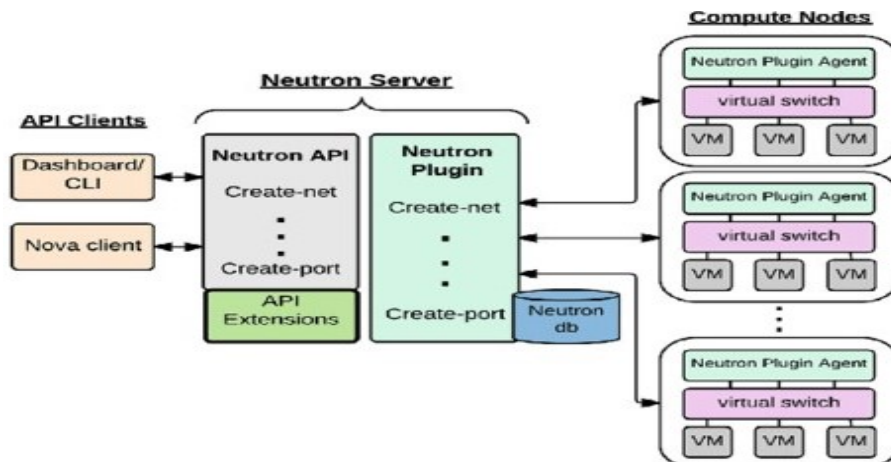


Figure 4: Estructura del módulo Neutron

Se puede ver un poco la interacción del módulo en la *Figura 4*, en la cual el flujo se representa como: comando u orden por parte del usuario desde el

cli o dashboard, que por medio de la API permite llamar a los binarios que crean la red o subnet, puertos y extensiones. Esto último, que es el servidor, interacciona con los nodos o instancias en sí, que crearán un virtual switch o interfaces de red en cuestión para poder conectarse y utilizar las funcionalidades.

Keystone

Se encarga de gestionar los usuarios en todos sus ámbitos: grupos, roles, permisos. En primera instancia, se puede decir que cumple dos funciones:

- **Gestión de usuarios:** Mantiene los permisos y su información de manera persistente en una base de datos.
- **Gestión de servicios:** Administra los servicios, guardando la ubicación de las APIs que permiten usar dichos servicios.

Este módulo es la base de la gestión y diseño de la topología que se verá más adelante, ya que los comandos introducidos por terminal deberán tener permisos. Si se intenta crear una red no va a dejar, de modo que lo que deberá hacerse es entrar en el panel (Dashboard) de OpenStack (con la IP de la interfaz de red HostOnly, tal y como se va a ver en un futuro). De este modo, se puede descargar en un apartado la denominada 'keystonerc-admin' que será un fichero con las directivas que permiten acceder como permiso de usuario administrador. Una vez descargado, con un scp se copia en la máquina remota (con la IP de destino de la interfaz HostOnly) y se hace un 'source keystonerc_admin' para poder repetir el comando, funcionando esta vez (esto se verá con capturas de pantalla cuando se explique la configuración total). Se logra esta funcionalidad:

- **Usuarios/Grupos:** Identidades digitales para usar OpenStack. Se cargan los permisos con el comando source. A su vez, los roles serán los permisos de estos grupos y usuarios en un tenant o proyecto dado.
- **Token:** Describe los recursos que pueden ser accedidos.

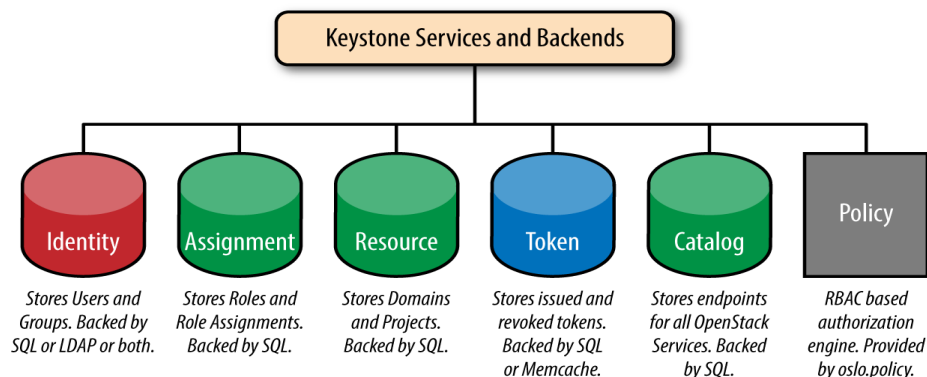


Figure 5: Servicios de Keystone

Todas estas funciones que tiene el módulo, se encuentran diversificadas en sub-módulos, siendo uno de ellos el de identidad en sí (para identificar a un usuario), otro para el almacenamiento de roles y recursos, así como uno para guardar los tokens de identificación y, finalmente, el catálogo (para habilitar que se usen los endpoints o API de Openstack en otros servicios). Todo esto aparece en la *Figura 5*.

Cinder

En primer lugar, al usar OpenStack se van a tener que distinguir dos tipos de almacenamiento posibles, a saber:

- **Efímero / Temporal (Nova):** Persiste mientras la instancia a la que se asocia siga vigente y no se elimine.
- **Persistente:**
 - **De bloque:** Cinder.
 - **De objetos:** Swift (un módulo que se explicará a continuación).
 - **De archivos (shared).**

Como puede verse, Cinder gestiona el almacenamiento persistente de bloque y, a su vez, es una copia de nova-volume en versiones actuales (tal y como se comentó brevemente en el módulo de nova). Emplea LVM y iSCSI como herramientas para acceder a los recursos almacenados.

La idea es que el volumen persistente creado se asocia a una instancia, pero no puede compartirse en dos, de modo que si quisiera pasar a formar parte de otra primero se debería suprimir de la instancia original. Para poder permitir esta funcionalidad compartida se tendría Manila, si bien no va a ser objeto de estudio y solamente se comenta.

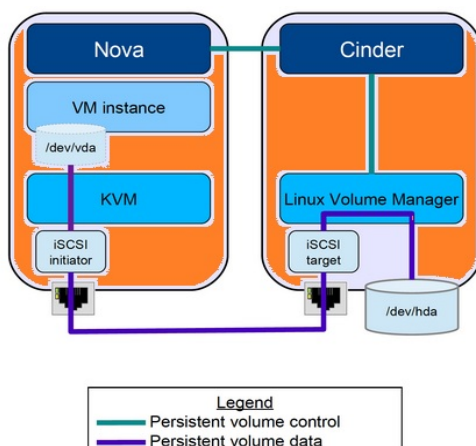


Figure 6: Arquitectura de Cinder

Este módulo se relaciona con el almacenamiento de una instancia. Por ello, su relación es con el módulo *Nova*, el cual emplea para acceder a las instancias y relacionarlas con la base de datos de volúmenes que contiene *Cinder*, todo ello por medio de *iSCSI*. Esto se ve en la *Figura 6*.

Swift

Se encarga de gestionar los objetos, entendiendo como tal a las entidades que almacenan información y que son únicas (en cuanto a que solamente hay una URL referenciando a dicho objeto para acceder al recurso).

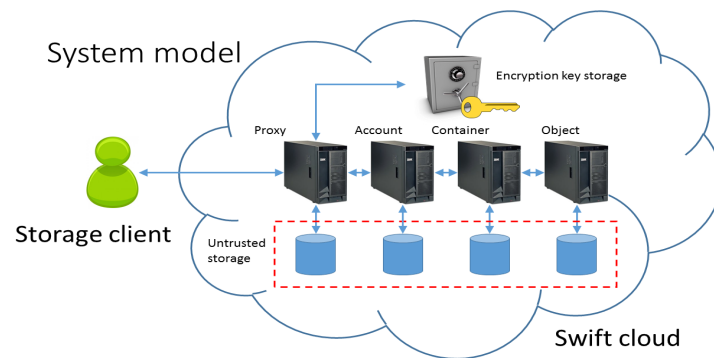


Figure 7: Arquitectura de Swift

Su uso es simple: un usuario cliente pretende hacer uso de un objeto, para lo cual se emplea un cliente de almacenamiento encargado de interaccionar con el ordenador en cuestión (ubicado en la nube de Swift, es decir, el conjunto de ordenadores con los objetos guardados). Una vez hecho esto, se necesitará una autenticación normalente, y en caso de poder acceder, se llegará al *Almacenamiento no confiado*, que aparece representado en la *Figura 7*.

Heat

La idea es que funciona como un template con un lenguaje (al estilo `.yaml` en administración de sistemas y ganglia) para poner en marcha un entorno cloud brevemente, definiendo en un archivo los hosts con sus características.

En la *Figura 8* se puede ver de qué manera la definición de un fichero `.yaml` que se carga en *Heat* se implementa automáticamente. Generalmente se ejecuta el comando de carga del fichero que lleva a cabo la llamada a la API y luego se cargan las variables de los proyectos de Openstack y los nodos. Finalmente, cuando el nodo sea el adecuado (aquel a modificar) se crean los puertos y se guardan los cambios, para poder desconectarse de la API.

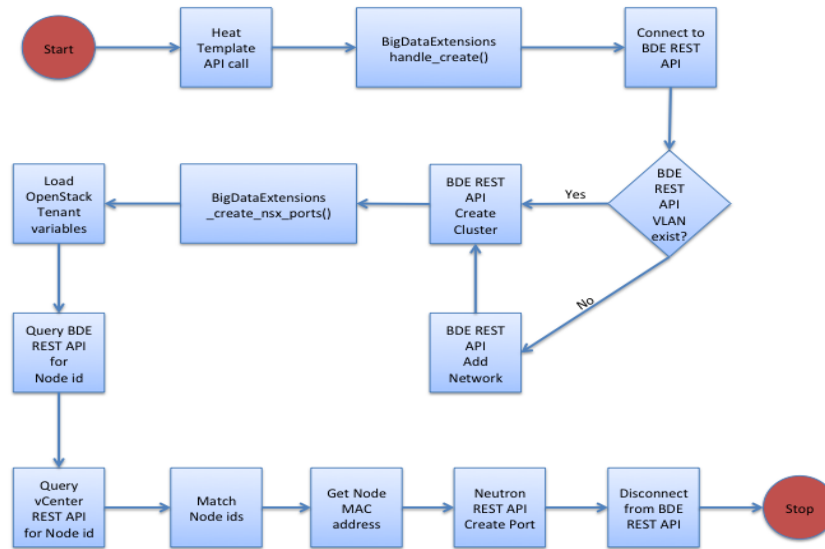


Figure 8: Arquitectura de Heat

2.2 Nagios

Sistema de monitorización de redes Open Source, muy usado actualmente a fin de conocer el desempeño de los sistemas en cuanto a una serie de parámetros que se podrán definir en la configuración. Vigila tanto los hosts que forman una red (hardware) como los servicios de cada host (software), siendo posible especificar distintos servicios para cada host. En cualquier caso, la herramienta se encarga de notificar al servidor (dirigido por el administrador de red) en caso de salir de los parámetros esperados los datos recogidos por la métrica. [9]

Se caracteriza por la versatilidad que aporta, ya que hay muchos servicios a monitorizar, además de que es posible crearlos de cero para parámetros más específicos. Además, las alertas para conocer el estado del sistema pueden ser notificadas por correo o sms, siendo necesario para ello definir unos dominios en los archivos de configuración (más adelante se explicará).

Las principales características que definen Nagios son:

- **Monitorización de servicios de red:** Protocolos disponibles:
 - **SMTP:** [*Protocolo para Transferencia Simple de Correo / Simple Mail Transfer Protocol*]. Se usa para intercambiar mensajes de correo entre dispositivos. Opera en el puerto 25 (no seguro) así como en el 465 (modo seguro), restando el 587 para los clientes alternativamente.

- **POP3:** [*Protocolo de Oficina de Correo / Post Office Protocol*] Protocolo empleado en clientes locales de correo para obtener los mensajes almacenados en un servidor remoto. Se emplea en el puerto 110 para la conexión no segura, así como el 995 para la cifrada.
 - **HTTP:** [*Protocolo de Transferencia de Hipertexto / HyperText Transfer Protocol*] Protocolo de comunicación que permite transmitir información a través de ficheros en la World Wide Web (WWW). El puerto definido para su uso es el 80 (Apache).
 - **ICMP:** [*Protocolo de Control de Mensajes de Internet / Internet Control Message Protocol*] Es utilizado para enviar mensajes de error, siendo el destinatario la dirección IP del origen del paquete.
 - **SNMP:** [*Protocolo Simple de Administración de Red / Simple Network Management Protocol*] Perteneciente a la capa de aplicación, facilita el intercambio de información de tipo administrativa, entre dispositivos de la red. Se emplea en los puertos 161 y 162 (TRAP).
- **Monitorización de recursos hardware:** Carga del CPU, logs, capacidad de discos, memoria. Es posible en Windows y Linux, aunque hay que instalar unos plugins (NRPE_NT, NSClient++).
 - **Monitorización remota:** Emplea túneles SSH, que pueden cifrarse.
 - **Diseño de plugins:** C++, Perl, Python, PHP, Bash, entre otros.
 - **Funcionalidad de red para distinguir la jerarquía:** Distinción entre hosts caídos o, por el contrario, solamente inaccesibles actualmente.
 - **Rotación automatizada del registro.**
 - **Visualización en tiempo real del estado del Sistema (Dashboard).**

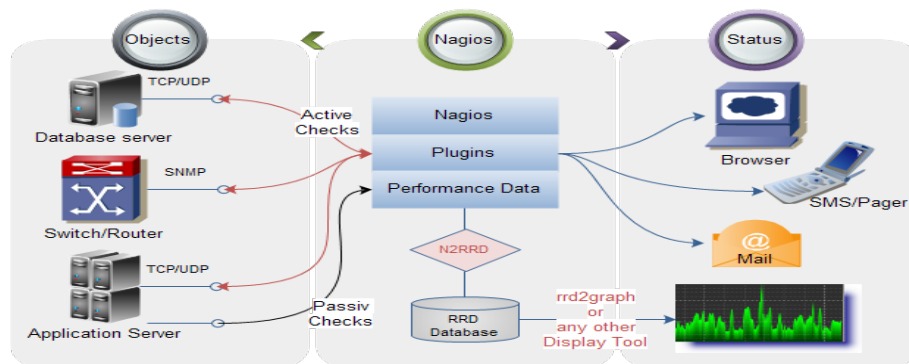


Figure 9: Arquitectura de Nagios

La Figura 9 muestra el componente central (*Nagios core*) que tiene los plugins y el server, siendo este el que se conecta con los objetos y con el estado del sistema. Este último sub-módulo (*Status*) representa las gráficas.

3 Marco práctico

El siguiente apartado va a conllevar la explicación de la parte central más importante a la hora de configurar la funcionalidad de la red buscada.

3.1 OpenStack - Infraestructura

En este apartado se va a explicar de qué manera se llevó a cabo la configuración de las instancias para poder desplegar la red privada del proyecto de una manera adecuada. Para ello, lo primero que hay que dejar claro es que se va a llevar a cabo una implementación *all_in_one*, lo cual significa que todos los módulos para mantener operativo OpenStack se van a encontrar en el servidor. Esta es la forma habitual de proceder para entornos de prueba, siendo la arquitectura aquella que se puede ver en la *Figura 10*.

Esta imagen representa las diferentes capas de abstracción de la implementación, a modo de pila (partiendo de la base que es el origen). Por tanto, puede verse como se parte de un ordenador (que sería el nativo) sobre el cual se trabaja en el Sistema Operativo de Windows para, por medio de un sistema de virtualización, desplegar una máquina virtual con Ubuntu 18.04, que será donde se implemente y configure OpenStack para la topología. Finalmente se configura Nagios en la red interna de OpenStack para monitorizar dicha topología. Las interfaces mostradas se explican en el *Apartado 3.1.3*, si bien es necesario darse cuenta del origen de las flechas que definen las interfaces así como su destino, ya que representan el funcionamiento (puertos de entrada y salida) que más tarde se explica.

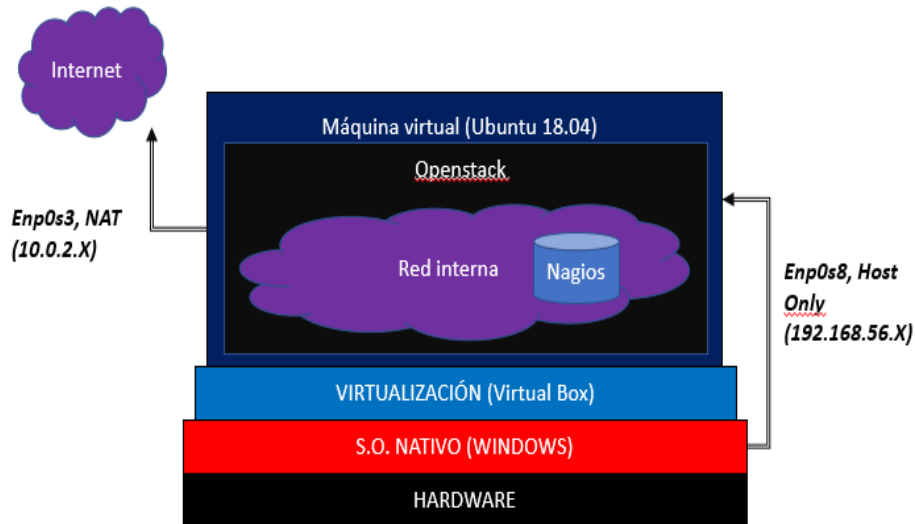


Figure 10: Arquitectura buscada

Sin embargo, hay alternativas con más flexibilidad a la hora de adaptarse al administrador, que tendrá más capacidad de alteración del entorno. Dichas alternativas se explicarán en el *Apartado 5*.

3.1.1 Elección de alternativas OpenStack

Una parte fundamental del proyecto es adecuarse a las 3 vertientes que ofrece OpenStack. Dichas alternativas son:

- **Devstack, Packstack, Microstack.**

Se puede resumir su operatibilidad y diferencias con la siguiente tabla:

Criterio evaluado	Packstack	Devstack	Microstack
Compatibilidad con Windows	Sí	Sí	Sí
Compatibilidad con Ubuntu	No	Sí	Sí
Recursos necesarios	Medios	Altos	Bajos

Considerando que la compatibilidad era un tema crucial para poder usar ubuntu, con su facilidad para configurar por medio de la terminal, se decidió a emplear la alternativa DevStack.

Es cierto que no es ligera en cuanto a recursos, siendo recomendado disponer a la máquina virtual contenedora con al menos 4GB para no ejecutar pobremente, si bien no supuso un impedimento considerando la compatibilidad, que era crucial.

3.1.2 Interfaces de red para el cluster: Motivos y diferencias entre los distintos tipos

A la hora de configurar la máquina virtual base, Virtual Box permite configurar las interfaces añadiendo, eliminando y seleccionando de qué tipo es cada una.

Antes de alcanzar una decisión en cuanto a su uso, es necesario saber qué se quiere hacer: la idea es tener una red virtual que conecte con el host (la máquina nativa) por medio de un solo punto, teniendo las instancias de la máquina virtualizada (máquina no nativa) acceso a internet y acceso entre ellas. Esto es, se quiere definir una red privada con acceso por medio de una interfaz que conecta la máquina nativa con la máquina virtual que desplegará Devstack, siendo dicha red un conjunto de máquinas virtuales, todas con acceso entre sí. A su vez, ya que esto se podría llegar a conseguir con diferentes configuraciones, cabe destacar que se pretende adaptar esto a cualquier red o, dicho de otro modo, se busca que el proyecto sea portable. De este modo, no se tiene que estar configurando todo cada vez que las redes físicas en las que se encuentra el pc nativo cambian.

En conclusión, de esta forma se pueden hacer backups o servidores redundantes

para balanceado de carga de una manera muy sencilla, simplemente exportando e instalando en la nueva localización, y ya funcionaría sin hacer más.

Ahora se explicarán las diferentes interfaces existentes [10]:

- **Bridge:** Configuración por defecto, ya que es la manera más sencilla de otorgar acceso a Internet a una máquina virtual.

Consiste en llevar a cabo una extensión de la red de área local (LAN), de modo que si está configurado un ordenador nativo para recibir por DHCP una IP de la red 192.168.1.x, la instancia virtual sería asignada con otra IP de esa misma red.

- **NAT:** [Network Address Translation] Interfaz pensada para solucionar el problema de la escasez de direcciones IP con la escalabilidad de dispositivos en los años 2000. De este modo, con una sola IP pública varios equipos pueden conectarse a Internet.

En las máquinas virtuales lo que ocurre es que se recibirá una dirección IP de un servidor DHCP virtual, sin embargo quien pide la IP será el firewall dentro de la aplicación de virtualización. De este modo, será el firewall y no la instancia quien se comunica con la red.

- **Host-Only:** Solamente se conecta esta interfaz con el ordenador nativo (anfitrión).

La máquina virtual está aislada de la red de área local, ya que la red de la máquina virtual está dentro del equipo nativo y es invisible para cualquier equipo de la red del equipo nativo.

- **Red Interna:** Se emplea para poder conectar máquinas virtuales entre sí, y es muy beneficioso para formar de una manera fácil diferentes redes privadas de máquinas virtuales. No permite conectarse con el Host.

En base a esto, la configuración de interfaces de red empleada para la máquina virtual en cuestión será:

- **Interfaz número 1:** Host-only, requerida para poder tener acceso por medio de virtualización a la máquina nativa, que será la encargada de ejecutar los comandos y conectarse (por medio del administrador) a los dashboards (cabe destacar que se va a emplear en las máquinas el mismo Sistema Operativo: un Ubuntu-18.04 modo server, sin interfaz gráfica, de ahí la necesidad de poder tener los paneles de control accesibles desde el host buscando en el navegador la URL la IP de la máquina virtual).
- **Interfaz número 2:** NAT, requerida para poder tener acceso a Internet en la máquina virtual y, consecutivamente, en la red de Devstack que se verá en un futuro.

También podría haberse llevado a cabo una configuración simple con un adaptador de tipo Bridge, pero no hubiera sido la red adaptable y portable. Además, el uso de varias interfaces de red permite una configuración más dinámica y más puntos de acceso a la máquina (ya que cada interfaz sería como un entry-point).

3.1.3 Configuración inicial

Una vez configuradas las interfaces, ya estarían activas correctamente (si bien luego se tendrá que programar su funcionalidad). Ahora bien, el próximo paso es llevar a cabo la instalación de Devstack. Para ello, se siguen los siguientes pasos en un script:

Example 1: Script de instalación

```
apt update -y && apt upgrade -y
init 6
sudo adduser -s /bin/bash -d /opt/stack -m stack
echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
su - stack
sudo apt install git -y
git clone https://git.openstack.org/openstack-dev/devstack
cd devstack
vim local.conf
./stack.sh
```

Una vez aplicado el último comando, se instalaría de acuerdo a las configuraciones del fichero local.conf (*Figura 11*), a saber:

```
[[local|localrc]] (1)

# Password for KeyStone, Database, RabbitMQ and Service
ADMIN_PASSWORD=sainzCRIS (2)
DATABASE_PASSWORD=$ADMIN_PASSWORD (3)
RABBIT_PASSWORD=$ADMIN_PASSWORD (4)
SERVICE_PASSWORD=$ADMIN_PASSWORD (5)

# Host IP - get your Server/VM IP address from ip addr command
HOST_IP=192.168.56.105 (6)
FLOATING_RANGE=192.168.56.224/27 (7)
FLAT_INTERFACE=enp0s8 (8)
```

Figure 11: Archivo de configuración de Devstack

Explicación de las líneas de código:

- **1.1:** Cabecera para el fichero.

- **l.2-5:** Autenticación en el dashboard para los módulos.
- **l.6:** Dirección IP privada de la máquina virtual con Devstack. Esta IP es la de la interfaz host-only, que se va a ver cómo se configuró en el siguiente apartado. Además, es la que se emplea en el navegador para acceder a los módulos. En definitiva, la base del proyecto.
- **l.7:** Rango flotante asignado para las direcciones IP externas con las cuales se conectará en el futuro a las máquinas virtuales de la red Devstack. Para evitar problemas de ambigüedad, se escogen con máscara 27 a partir de la 224, lo cual significa que las últimas 32 IP's que van del 224 al 255 son para estas máquinas virtuales.
- **l.8:** Asignación de la interfaz `enp0s8` como red externa o pool para las máquinas virtuales. Se puede decir que con esta directiva se especifica que la red que tiene `enp0s8` es el pool, y luego con la directiva `FLOATING_RANGE` se limita dicha pool.

En el apartado siguiente se verá de qué manera se han programado las interfaces, ya que de momento se han activado los dos tipos, pero ahora queda darles la configuración adecuada, que hará que este fichero de configuración se entienda mejor.

- Podría haberse puesto una línea como esta: `install_heat = yes`, la cual habría instalado el módulo en cuestión, ya que por defecto no se considera en el paquete. Al no pretender usarse, no se instaló.

3.1.4 Programación de interfaces de red

Tal y como pudo verse en el *Apartado 3.1.2*, se han definido dos interfaces que sirven para poder construir nuestra topología adecuadamente.

La idea ahora es poder configurar dichas interfaces correctamente. Para ello, es necesario entrar en el archivo `/etc/network/interfaces` y configurarlo tal y como se ve en la *Figura 12*:

```
auto lo (1)
iface lo inet loopback (2)

auto enp0s3 (3)
iface enp0s3 inet dhcp (4)

auto enp0s8 (5)
iface enp0s8 inet static (6)
address 192.168.56.105 (7)
netmask 255.255.255.0 (8)
broadcast 192.168.56.255 (9)
network 192.168.56.0 (10)
```

Figure 12: Configuración de interfaces en el sistema

Su explicación es:

- **1.1-2:** Definición y configuración de la interfaz loopback (127.0.0.1):
 - **1.1:** definición explícita.
 - **1.2:** definición como resolución recursiva (el mismo host, loopback).
- **1.3-4:** Definición y configuración de la interfaz *enp0s3* o NAT:
 - **1.3:** definición explícita.
 - **1.4:** definición de la interfaz como resolución por dhcp.
- **1.5-10:** Definición y configuración de la interfaz *enp0s8* o *Host Only*:
 - **1.5:** definición explícita.
 - **1.6:** definición de la interfaz como estática.
 - **1.7:** dirección IP estática asignada a la interfaz *enp0s8*.
 - **1.8:** máscara de red de la intra-net que se está diseñando, siendo en este caso de /24.
 - **1.9:** Dirección de broadcast o reenvío a todos los elementos de la red interna. Se trata, por convenio, de la última dirección IP de la interfaz, de modo que si la red tratada es *192.168.56.x*, que es aquella asignada por defecto por *Virtual Box*, se trataría de la red *192.168.56* con dispositivos del *192.168.56.0* al *192.168.56.255*, tal y como la máscara indica.
 - **1.10:** red dada, tal y como se ha visto en el anterior punto.
- *Importante reiniciar el servicio para coger la nueva configuración. Se puede hacer un 'systemctl reload/restart networking/network.service' o directamente un 'init 6' para reiniciar la instancia y volver a cargar en el booting time.*

Se puede ver correctamente su funcionamiento en la *Figura 13* (aunque hay interfaces que más tarde se explican y que de momento no aparecen):

```
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
inet6 fe80::a00:27ff:fea2:ed5f prefixlen 64 scopeid 0x20<link>
ether 08:00:27:a2:ed:5f txqueuelen 1000 (Ethernet)
RX packets 32 bytes 4960 (4.9 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 51 bytes 5393 (5.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.56.105 netmask 255.255.255.0 broadcast 192.168.56.255
inet6 fe80::a00:27ff:fe9e:4e79 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:9e:4e:79 txqueuelen 1000 (Ethernet)
RX packets 4 bytes 336 (336.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 12 bytes 936 (936.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
```

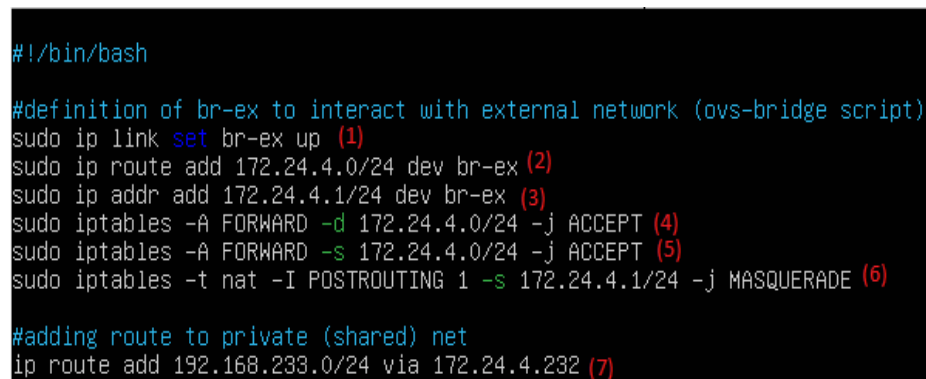
Figure 13: Interfaces

3.1.5 BR-EX: Reenvío de paquetes

Se debe llevar a cabo una configuración interna de las interfaces virtuales creadas por la instalación de Openstack. Para ello, entiéndase la diferencia entre las 3 interfaces creadas:

- **br-ex:** Interfaz que conecta la red interna de Openstack con el exterior (hardware nativo y red nativa). Crea la conexión entre la interfaz física y la red externa de Openstack (pero interna, al fin y al cabo, al ser dentro de la máquina virtual, en Openstack).
- **br-int:** Interfaz asociada directamente a la instancia, la cual permite recibir tráfico del exterior y reenviarlo desde la propia instancia a este mismo. Se trata de un método de integración para mapear puertos que permitan redirigir el tráfico.
- **br-tun:** Se trata el método de tunneling para poder permitir el flujo entre diferentes instancias pertenecientes a un mismo proyecto.

Una vez hecho esto, es necesario llevar a cabo una configuración inicial, en la que se tenga en cuenta la redirección de flujos y configuración de interfaces. La primera parte de la configuración se puede ver en la *Figura 14*:



```
#!/bin/bash

#definition of br-ex to interact with external network (ovs-bridge script)
sudo ip link set br-ex up (1)
sudo ip route add 172.24.4.0/24 dev br-ex (2)
sudo ip addr add 172.24.4.1/24 dev br-ex (3)
sudo iptables -A FORWARD -d 172.24.4.0/24 -j ACCEPT (4)
sudo iptables -A FORWARD -s 172.24.4.0/24 -j ACCEPT (5)
sudo iptables -t nat -I POSTROUTING 1 -s 172.24.4.1/24 -j MASQUERADE (6)

#adding route to private (shared) net
ip route add 192.168.233.0/24 via 172.24.4.232 (7)
```

Figure 14: Configuración de puertos virtuales

La explicación es la siguiente:

- **1.1:** con esta línea se lleva a cabo el linkado de la interfaz br-ex, poniéndola en modo activo o corriendo (interfaz en modo up, *comando 'ifup'*)
- **1.2:** pretende llevar a cabo la adición de una ruta, siendo esta ruta la que se identifique con la interfaz *br-ex*. Dicha ruta será la IP *172.24.4.0* con máscara */24*, que, tal como se ha visto, es la IP externa por defecto al usar Openstack. Por lo tanto, esta línea mapea la red externa con la interfaz virtual que permite conectar la red-externa con Internet.

- **1.3:** permite la adición de una IP que será la *172.24.4.1* con máscara de red */24* asociada también a la interfaz *br-ex*. A diferencia de la anterior línea, que pretendía asignar una nueva ruta por defecto para redirigir el tráfico hacia el exterior, en este caso se pretende simular el *router*, tal y como puede verse en la IP dada. En cualquier caso, esta línea *no define una ruta por defecto, más bien un dispositivo con una IP dada asignada*.
- **1.4:** configuración dada para poder aceptar el tráfico cuyo destino es hacia la red *172.24.4.0* con máscara de red */24*.
- **1.5:** configuración dada para poder aceptar el tráfico cuyo origen es la red *172.24.4.0* con máscara de red */24*.

En resumen, las dos últimas directivas pretenden que el entorno sea bidireccional en cuanto a lo que respecta la red interna dada (*172.24.4.0*), de modo que no solamente pueda enviarse tráfico desde esta red al exterior (Internet) o al revés.

- **1.6:** configuración dada para que se permita un correcto funcionamiento de la interfaz *NAT* ya vista. Para ello, es interesante ver el parámetro *-t*, que sirve para definir el tipo de paquetes que se van a tener en cuenta en esta regla, siendo en este caso los relativos a *NAT*. Viene acompañado del parámetro *-I*, que sirve para definir la opción *POSTROUTING*, esto es, que se aplica a *NAT* cuando son paquetes que van a salir a internet, permitiéndose su modificación en cuanto a la traducción que requieren. Esto se puede ver en el parámetro *-j*, que es el que hace la traducción/masquerade. Finalmente, el parámetro *-s* indica que esta regla se aplica a los paquetes que, además de cumplir las demás características, tienen como origen el dispositivo a modo de router, *172.24.4.1/24*.

De este modo, cuando se envía tráfico que no tiene una ruta definida en el routing table, llega por defecto al router, que al ver la dirección de origen hace un forwarding a la interfaz virtual *br-ex*, que conecta con la interfaz física externa del ordenador nativo. Así es como tiene lugar el direccionamiento.

- **1.7:** se trata de la adición de una ruta por defecto, que quiere decir que cualquier paquete que vaya dirigido a la red *192.168.233.0/24* (que más tarde se definirá como la red privada de Openstack), es enviado al dispositivo *172.24.4.232*, que será la dirección desde la cual se tiene visibilidad a la red interna, siendo en este caso el propio router que se verá en el apartado de Openstack. De este modo, si se quiere acceder a la red interna de OpenStack y se viene de Internet (exterior) pasando por *br-ex*, se redirige el tráfico por el router con IP externa dada en esta línea.

Ahora bien, esta configuración no es persistente, en el sentido de que los comandos no permanecen fijos en el sistema en el caso de ejecutarse una vez, algo parecido a cuando definimos una variable por *command line* en vez de en

el fichero `.bashrc`, conocido por ser el inherente a los alias.

Precisamente para hacerlos persistentes, se han guardado en un script (tal y como puede verse en la línea de `#!/bin/bash` que hay en la *Figura 14*). Dicho script se ejecuta en todo momento al rebootear el sistema, para lo cual se ha elegido la alternativa de hacer uso del *init.d* que es una serie de ficheros que ejecutan en el arranque automáticamente. La configuración dada puede verse en la *Figura 15* y consta de:

- Situar el fichero a automatizar (script) en el directorio *init.d*
- Actualización de la tabla de ejecutables(*update-rc.d*)
- **Importante:** es necesario que sea un ejecutable para poder funcionar en el *init.d*, de modo que se debe aplicar antes de terminar el comando `chmod +x script.sh`

Quedando de esta manera:

```
root@devstack:/opt/stack# cd /etc/rc0.d/
root@devstack:/etc/rc0.d# ls
K01apache2      K01ipvsadm      K01mysql        K01radvd
K01apache-htcacheclean K01irqbalance  K01networking   K01rsyslog
K01conntrackd   K01iscsid       K01open-iscsi   K01tgt
K01cryptdisks   K01keepalived  K01openvswitch-switch K01unattended-upgrades
K01cryptdisks-early K01lvm2-lvmetad K01plymouth     K01uuid
K01ebtables     K01lvm2-lvmpolld K01postgresql   K01uuuigi
K01haproxy      K01memcached    K01rabbitmq-server script.sh
root@devstack:/etc/rc0.d# mv script.sh /etc/init.d/script.sh
root@devstack:/etc/rc0.d# sudo update-rc.d script.sh defaults
root@devstack:/etc/rc0.d#
```

Figure 15: Configuración del *init.d*

Otra característica importante es permitir que el script ejecute como 'root' sin necesidad de pedir permiso parando su ejecución. Para ello se descomentaba esta línea del fichero `/etc/sudoers`.

Example 2: Configuración sudoers

```
#cristian ALL=(ALL)NOPASSWD: /etc/init.d/script.sh
#stack ALL=(ALL)NOPASSWD: /etc/init.d/script.sh
```

Ahora se verán las últimas configuraciones necesarias para terminar este apartado:

```
[ml2_type_vlan]
network_vlan_ranges = public,physnet_em1
```

Figure 16: VLAN nueva: `physnet_em1`

Como puede verse en la *Figura 16*, se crea una nueva VLAN para poder trabajar en esa como base del entorno Openstack.

Otra configuración necesaria es la vista en la *Figura 17*:

```
[ovs]
datapath_type = system
bridge_mappings = public:br-ex
tunnel_bridge = br-tun
local_ip = 192.168.56.105
```

Figure 17: Tunneling y mapeado de la red externa

De acuerdo a la *Figura 17*, el mapeado del puente que une la red externa de Openstack con la pública nativa tiene lugar. También se define el mapeo de la interfaz para hacer tunneling entre instancias, así como la IP local de la interfaz `enp0s8` que es el ordenador virtual con Openstack.

Por defecto, esta configuración debería ser suficiente, si bien en el caso específico tratado hubo que activar la interfaz (cuya configuración ya había sido definida) de la siguiente manera, creando su puerto [11][12]:

Example 3: Configuración ovs (puerto)

```
ovs-vsctl add-br br-ex
ifconfig enp0s8 && ovs-vsctl add-port br-ex enp0s8
```

Finalmente, es necesario asegurarse de que las rutas están correctamente creadas, para poder funcionar junto a la configuración dada:

```
root@devstack:~# route
Tabla de rutas IP del núcleo
Destino      Pasarela      Genmask      Indic Métric Ref      Uso Interfaz
default      _gateway      0.0.0.0      UG    0      0      0 enp0s3
default      _gateway      0.0.0.0      UG    100    0      0 enp0s3
10.0.2.0     0.0.0.0      255.255.255.0 U      0      0      0 enp0s3
_gateway     0.0.0.0      255.255.255.255 UH    100    0      0 enp0s3
172.24.4.0   0.0.0.0      255.255.255.0 U      0      0      0 br-ex
172.24.4.0   0.0.0.0      255.255.255.0 U      0      0      0 br-ex
192.168.56.0 0.0.0.0      255.255.255.0 U      0      0      0 enp0s8
192.168.122.0 0.0.0.0     255.255.255.0 U      0      0      0 virbr0
192.168.233.0 172.24.4.232 255.255.255.0 UG    0      0      0 br-ex
```

Figure 18: Tabla de rutas de la red core

De acuerdo a la *Figura 18* puede verse como existen:

- rutas hacia el gateway por defecto para el tráfico NAT: **enp0s3**
- rutas a través de la red con IP `172.24.4.0/24` (red pública de Openstack) así como por la red privada `192.168.233.0/24`: **br-ex**

- rutas a través de la interfaz Host Only (192.168.56.0/24): **enp0s8**
- la ruta virbr0 es creada por defecto al instalar Openstack, siendo esta una interfaz de red, como puede verse en la *Figura 19*.

```
virbr0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 192.168.122.1 netmask 255.255.255.0 broadcast 192.168.122.255
    ether 52:54:00:a6:ff:47 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 19: Interfaz de red virbr0

Una serie de configuraciones que se han obviado hasta ahora son:

Example 4: Configuración ml2 extendida

```
[ml2]
type_drivers=flat,vlan,vxlan
mechanism_drivers = openvswitch
[ml2_type_flat]
flat_networks = physnet1,physnet2
flat_networks = *
[ovs]
integration_bridge=br-int
```

En esta configuración se puede ver como se configura el tipo de redes posibles (drivers o puntos de entrada dados, **1.2**), el tipo de mecanismo de integración dado para Openstack (que podría ser bridge, pero se configuró como openvswitch para poder operar con las 3 interfaces br-ex,br-int y br-tun, **1.3**), así como finalmente el tipo de redes físicas que permiten crear una red de tipo plana (**1.5**), siendo también posible emplear la **1.6** para permitir cualquiera. Finalmente, en la última línea se define la interfaz *br-int*. **[12][13]**

Todo esto se puede llevar a cabo en CentOS de una manera mucho más intuitiva, con la siguiente configuración en el directorio */etc/sys/network-scripts/*, donde cada fichero es una interfaz:

Example 5: Configuración interfaz br-ex (ifcfg-br-ex)

```
DEVICE=br-ex
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
```

Example 6: Configuración interfaz enp0s3 (ifcfg-enp0s3)

```
DEVICE=enp0s3
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=172.24.4.1
ONBOOT=yes
```

Lo bueno de configurar una intra-net con el modo bridge es que directamente se puede redirigir, como se ve en las tablas superiores, el tráfico por la interfaz *br-ex*, ya que en este caso se aprecia que se pretende mapear la configuración de *enp0s3* en *br-ex*, de modo que sea este último el que, actuando a modo de puente, permita el tráfico. Funcionaría igual que la configuración original, si bien no requiere tanto a la hora de usar las interfaces de red de Openstack.

3.1.6 Redirección de flujo en Windows

En este punto, la redirección de flujo dentro de la máquina virtual (con todo lo que conlleva en cuanto a configuraciones) ya se ha llevado a cabo correctamente.

Sin embargo, si bien es cierto que se tendrá esta máquina host de Openstack corriendo, realmente va a actuar a modo de servidor. Esto quiere decir que la idea es no necesitarla para trabajar. Es por ello que una de las interfaces virtuales de red es la de conexión al host, para permitir ese flujo de datos.

Ahora va a ser necesario redirigir el tráfico (tanto de la red interna o privada como externa o pública) hacia la máquina virtual (aquella con IP *192.168.56.x*.)

Inicialmente se tiene una tabla de rutas como la de la *Figura 20*:



```
IPv4 Tabla de enrutamiento
=====
Rutas activas:
Destino de red      Máscara de red      Puerta de enlace      Interfaz      Métrica
-----
0.0.0.0             0.0.0.0             192.168.1.1           192.168.1.9    25
127.0.0.0           255.0.0.0           En vínculo            127.0.0.1     331
127.0.0.1           255.255.255.255     En vínculo            127.0.0.1     331
127.255.255.255     255.255.255.255     En vínculo            127.0.0.1     331
192.168.1.0         255.255.255.0       En vínculo            192.168.1.9    281
192.168.1.9         255.255.255.255     En vínculo            192.168.1.9    281
192.168.1.255       255.255.255.255     En vínculo            192.168.1.9    281
192.168.56.0        255.255.255.0       En vínculo            192.168.56.1    281
192.168.56.1        255.255.255.255     En vínculo            192.168.56.1    281
192.168.56.255      255.255.255.255     En vínculo            192.168.56.1    281
224.0.0.0           240.0.0.0           En vínculo            127.0.0.1     331
224.0.0.0           240.0.0.0           En vínculo            192.168.56.1    281
224.0.0.0           240.0.0.0           En vínculo            192.168.1.9     281
255.255.255.255     255.255.255.255     En vínculo            127.0.0.1     331
255.255.255.255     255.255.255.255     En vínculo            192.168.56.1    281
255.255.255.255     255.255.255.255     En vínculo            192.168.1.9     281
=====
Rutas persistentes:
Ninguno
```

Figure 20: Comando 'route print'

Ahora bien, se crean las dos rutas con el modo persistente (*argumento '-p'*), siendo los comandos los siguientes:

```
Example 7: Redirección de flujo a la instancia virtual
route -p ADD 172.24.4.0 MASK 255.255.255.0 192.168.56.105
route -p ADD 192.168.233.0 MASK 255.255.255.0 192.168.56.105
```

Quedando la tabla de rutas de la siguiente manera en la parte de rutas persistentes, tal y como se ve en la *Figura 21*:

Rutas persistentes:				
Dirección de red	Máscara de red	Dirección de puerta de enlace	Métrica	
192.168.233.0	255.255.255.0	192.168.56.105	1	
172.24.4.0	255.255.255.0	192.168.56.105	1	

Figure 21: Rutas persistentes hacia la instancia virtual

Una vez definidas, queda claro que si se intenta enviar un paquete a cualquiera de estas redes (recuérdese que se define una máscara, así que cualquier IP en el rango de la red valdría) se rediriría por la ruta estática permanente. Esto se puede ver con facilidad en la *Figura 22*:

```
C:\Windows\system32>tracert 172.24.4.188

Traza a 172.24.4.188 sobre caminos de 30 saltos como máximo.

  1    1 ms    1 ms    1 ms    192.168.1.1
  2   23 ms   14 ms   21 ms   10.194.120.1
  3   12 ms   12 ms   11 ms   10.254.36.81
  4   20 ms   25 ms   24 ms   10.255.101.169
^C
C:\Windows\system32>route -p ADD 172.24.4.0 MASK 255.255.255.0 192.168.56.105
Correcto

C:\Windows\system32>tracert 172.24.4.188

Traza a 172.24.4.188 sobre caminos de 30 saltos como máximo.

  1    <1 ms   <1 ms   <1 ms   192.168.56.105
  2    <1 ms   <1 ms   <1 ms   172.24.4.188
  3     1 ms   <1 ms   <1 ms   172.24.4.188

Traza completa.
```

Figure 22: Tracking de rutas con 'tracert'

De acuerdo a esta *Figura 22*, queda claro que, inicialmente, se está encontrando una ruta para poder redirigir, o al menos eso se puede pensar. En verdad, al no haber una ruta no está encontrándola, lo cual lleva a buscar por medio de los DNS que se tienen configurados, a ver si se puede resolver la IP en alguna tabla de rutas o archivo DNS configurado previamente. Es por ello que va saltando entre redes con IPs muy distintas, sin llegar a alcanzar nuestra IP buscada.

Una vez añadida la ruta, solamente le toma dos saltos llegar, uno para poder pasar del ordenador nativo a la interfaz virtual que conecta con la instancia hosteadora de Openstack, y la otra para, una vez dentro, encontrar la instancia virtual de Openstack con la IP dada (ya que *172.24.4.188* se corresponde con una instancia de la topología que se verá en el próximo apartado 3.1.7)

Más tarde, se verá de qué manera esto sirve para, también poder conectar a una instancia desde el host nativo, por medio de las llaves creadas en el apartado 3.1.7

3.1.7 Definición de topología OpenStack

En este apartado se va a llevar a cabo la explicación de los diferentes apartados de Openstack, así como una pequeña guía a lo largo de su configuración

En primer lugar debe quedar claro que se va a emplear CLI (Commando Line Interface) para crear la arquitectura [14], de modo que va a ser necesario disponer de permisos. Esto si se loguea uno desde el dashboard es sencillo, pero en el caso del CLI requiere cargar un binario, esto es, un fichero que dota de permisos a la sesión activa de la terminal. Para ello, se debe loguear en *192.168.56.105/dashboard* y arriba a la derecha descargar el fichero admin (key). Una vez hecho esto, se sube a una página para poder descargarse en la instancia con facilidad. Puede verse en la *Figura 23*, junto al error inicial de no poder cargar los comandos por permisos.

```
root@devstack:~# openstack network list
/usr/lib/python3/dist-packages/secretstorage/dhcrypto.py:15: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/usr/lib/python3/dist-packages/secretstorage/util.py:19: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
Missing value auth-url required for auth plugin password
root@devstack:~# ls
root@devstack:~# ls /home/cristian/
root@devstack:~# wget https://file.io/mjsajeIJcaul
--2021-03-20 18:49:38-- https://file.io/mjsajeIJcaul
Resolviendo file.io (file.io)... 52.22.39.17, 23.20.132.104
Conectando con file.io (file.io)[52.22.39.17]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: no especificado [application/octet-stream]
Guardando como: "mjsajeIJcaul"

mjsajeIJcaul [ <-> ] 1,90K --.-KB/s en 0s
2021-03-20 18:49:39 (126 MB/s) - "mjsajeIJcaul" guardado [1941]

root@devstack:~# mv
.bash_history .cache/ .local/ .profile
.bashrc .gnupg/ mjsajeIJcaul
root@devstack:~# mv mjsajeIJcaul key
root@devstack:~# source key
Please enter your OpenStack Password for project admin as user admin:
root@devstack:~#
```

Figure 23: Fichero source de Openstack

Una vez hecho esto, lo primero que deberá hacerse será crear las redes, tanto

externa (pública) como privada (interna) del entorno Openstack (junto a, por supuesto, sus subredes para delimitar los rangos de IPs):

Example 8: Definición de redes y subredes

```
openstack network create private
openstack subnet create --network private \
  --allocation-pool start=192.168.233.100,end=192.168.233.254 \
  --dns-nameserver 8.8.8.8 \
  --subnet-range 192.168.233.0/24 \
  private_subnet

openstack network create \
  --provider-network-type flat \
  --provider-physical-network extnet \
  --external \
  public
openstack subnet create --network public \
  --allocation-pool start=172.24.4.1,end=172.24.4.254 \
  --no-dhcp \
  --subnet-range 172.24.4.0/24 public_subnet
```

Una vez hecho esto, ya se tendrían las redes con sus ámbitos IPs, de modo que ahora sería necesario crear un router que permita direccionar entre ambas redes y hacia el exterior (sin dicho router, el tráfico interno no puede llegar a internet). Para ello, el router tendrá que conectar con 2 nuevas interfaces de red que unan ambas subredes creadas:

Example 9: Definición de router e interfaces

```
openstack router create --no-ha router1
openstack router set --external-gateway public router1
openstack router add subnet router1 private_subnet
```

Ahora bien, antes de crear una instancia (con sus respectivos pasos, que ya se verán) es necesario definir un grupo de seguridad con permisos de puertos, esto es, con una serie de puertos abiertos para poder llevar a cabo de manera correcto el flujo de datos. Para ello:

Example 10: Definición de grupo de seguridad y filtro de puertos

```
openstack security group create basic --description "Allow base ports"
openstack security group rule create --protocol TCP --dst-port 22 \
  --remote-ip 0.0.0.0/0 basic
openstack security group rule create --protocol TCP --dst-port 80 \
  --remote-ip 0.0.0.0/0 basic
openstack security group rule create --protocol TCP --dst-port 443 \
  --remote-ip 0.0.0.0/0 basic
openstack security group rule create --protocol ICMP \
```

```
| --remote-ip 0.0.0.0/0 basic
```

Como puede verse se han filtrado las comunicaciones por medio de la apertura o concesión de permisos a los puertos 22 (SSH), 80 (HTTP), 443 (HTTPS) y, finalmente, ICMP (para poder hacer *PING*).

Ahora se creará una llave para, posteriormente (al igual que el grupo de seguridad) poder asignar a las instancias y conectar por medio de ella al hacer SSH. Para ello, se genera la key y se importa en Openstack:

Example 11: Definición de llave de acceso mediante SSH

```
ssh-keygen -q -N ""  
openstack keypair create --public-key=~/.ssh/id_rsa.pub llave_vm
```

Ahora solamente faltaría crear una instancia. Para ello es necesario, primero, tener una imagen base de Sistema Operativo para asociar a la instancia. Esto va a constar de descargar (por medio del *wget*) dicha imagen de un repositorio como el oficial de Ubuntu. Después importar la imagen a Openstack. Quedaría:

Example 12: Importación de imagen de S.O. a Openstack

```
wget <url_de_la_imagen>  
openstack image create <nombre> \  
--file <ruta_imagen> --min-ram <ram_minima> \  
--min-disk <min_discos_virtuales> --disk-format qcow2
```

Esto se hizo para dos S.O: Ubuntu 18.04 server y cirros (el que viene por defecto en Openstack). Sin embargo, esto aún solamente es tener la imagen disponible en Openstack, ya que falta crear la instancia:

Example 13: Creación de instancia

```
openstack server create <nombre> \  
net-id=<red_privada> --flavor <nombre_flavor> \  
--image <nombre_imagen> --key-name <nombre_llave> \  
--security-group <nombre_grupo_seguridad>
```

El atributo de flavor hace referencia al sabor (que son las capacidades o limitaciones de una imagen, esto es, RAM, discos, etc.) y no tiene por qué ponerse, si bien se puede crear uno de esta manera:

Example 14: Definición de sabor (flavor)

```
openstack flavor create <nombre> --vcpus 1 --ram 128 --disk 1 --id 8
```

Solamente quedaría asignar una IP flotante (que es del rango de la red pública creada) para poder salir a Internet, asignándola a la instancia creada anteriormente con el flavour y la imagen dados:

Example 15: Definición de IP flotante asociada a instancia

```
openstack floating ip create <nombre_red_externa>
openstack server add floating ip <instancia> <ip_del_anterior_comando>
```

Si bien no se han ido poniendo los outputs de todos los comandos (ya que sería mucho overhead), finalmente se va a demostrar tanto para la instancia de cirros (*Figura 24*) como la de ubuntu (*Figura 25*), que funciona correctamente (conectando con la IP externa flotante asignada):

```
C:\Windows\system32>ssh cirros@172.24.4.188
cirros@172.24.4.188's password:
$ pwd
/home/cirros
$ exit
Connection to 172.24.4.188 closed.

C:\Windows\system32>ssh cirros@192.168.233.100
cirros@192.168.233.100's password:
$ pwd
/home/cirros
$
```

Figure 24: Instancia de cirros con IP flotante y privada

```
root@devstack:~# ssh -i /opt/stack/llave_vm ubuntu@172.24.4.179
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-139-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Thu Mar 25 11:46:48 UTC 2021

System load:  0.41               Processes:    80
Usage of /:   22.2% of 4.67GB    Users logged in: 0
Memory usage: 24%               IP address for ens3: 192.168.233.130
Swap usage:   0%

0 packages can be updated.
0 of these updates are security updates.

Last login: Thu Mar 25 11:33:21 2021 from 172.24.4.1
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ubuntu-host:~$
```

Figure 25: Instancia de Ubuntu con IP flotante

3.1.8 Prueba de funcionamiento con PuTTY

En este apartado se explicará de qué manera se tienen que tener configuradas las llaves, que se han creado en Openstack. Estas sirven para poder conectar a las instancias por medio de *SSH* y es así como se han configurado las instancias en el apartado anterior.

La idea es muy breve, ya que todas las configuraciones se tienen adecuadamente, de modo que este apartado solamente son un par de apuntes:

- **En primer lugar**, es necesario que la llave se encuentre en el ordenador nativo. Para ello ver la *Figura 26*.
- **En segundo lugar**, se debe tener cuidado con los permisos, ya que pueden ser muy abiertos y entrar en conflicto con aquellos de Openstack. Podría suceder que aparezca el error de la *Figura 27*, el cual se soluciona haciendo un `chmod 640 llave`, en el caso de ser la llave privada o, si se ha equivocado el usuario y ha importado la pública, importando la privada. Es muy fácil que aparezca error con la privada, así que recuérdese esto, si bien en la *Figura 27* se muestra el error de trabajar con la pública cuando el pc nativo está conectado con Openstack y, efectivamente, forman uno que es indistinguible.

```
C:\Users\cris>ssh ubuntu@172.24.4.179
ubuntu@172.24.4.179: Permission denied (publickey).

C:\Users\cris>scp root@192.168.56.105:/opt/stack/llave vm C:\Users\cris\Desktop\tfg
root@192.168.56.105's password:
llave_vm
```

Figure 26: Importación de la llave al pc nativo

```
C:\Users\cris>ssh -i C:\Users\cris\Desktop\tfg\llave vm.pub ubuntu@172.24.4.179
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                WARNING: UNPROTECTED PRIVATE KEY FILE!                @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions for 'C:\\Users\\cris\\Desktop\\tf\\llave_vm.pub' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "C:\\Users\\cris\\Desktop\\tf\\llave_vm.pub": bad permissions
ubuntu@172.24.4.179: Permission denied (publickey).
```

Figure 27: Error de permisos en la llave importada

Ahora bien, una vez se tenga esto en cuenta, si bien es cierto que se puede conectar por medio de cualquier tipo de terminal, lo más cómodo es emplear PuTTY. Para ello, una vez instalado, se entra en el programa PuTTY Gen, se clicka en load y se pone la clave privada importada. Finalmente se le da a generar y se consigue dicha llave en el formato .ppk de putty (putty private key). Ahora se va al programa PuTTY original, y se pone en Host la IP *172.24.4.x* (referente a la instancia dada) y en SSH Auth se explora y busca la llave de formato .ppk recientemente generada. Con todo ello debería dejar conectar, tal y como se ve en la *Figura 28*.

```
ubuntu@ubuntu-host: ~  
* Using username "ubuntu".  
* Authenticating with public key "imported-openssh-key"  
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-139-generic x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/advantage  
  
System information as of Thu Mar 25 23:36:49 UTC 2021  
  
System load:  0.22               Processes:            80  
Usage of /:   22.2% of 4.67GB    Users logged in:     0  
Memory usage: 24%               IP address for ens3: 192.168.233.130  
Swap usage:   0%  
  
* Introducing self-healing high availability clusters in MicroK8s.  
Simple, hardened, Kubernetes for production, from RaspberryPi to DC.  
  
https://microk8s.io/high-availability  
  
0 packages can be updated.  
0 of these updates are security updates.  
  
Last login: Thu Mar 25 23:11:30 2021 from 172.24.4.1  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
ubuntu@ubuntu-host:~$
```

Figure 28: Prueba de PuTTY

3.1.9 Problemas comunes

A la hora de trabajar con Openstack se han encontrado una serie de problemas menores, que es interesante comentar para poder paliar con facilidad en un futuro. No tienen nada que ver con la configuración, ya que si se comenta en profundidad las pruebas y configuraciones de los archivos ml2 o scripts vistos con anterioridad, se extendería mucho el documento. Más bien, tienen que ver con la propia funcionalidad.

Una vez dicho esto, los errores a comentar son estos:

- **Fallo en los módulos:** En algunos casos, después de iniciar la máquina de Openstack se pudo ver como, si bien al principio arrancaban bien los servicios, terminaban después de un rato cayéndose por sí solos. Esto se veía claramente en dos errores que, analizados con lógica, hicieron que se pensase en una posible recaída de los servicios que hostean los módulos, a saber:

- **Keystone caído:** Se entra en el pc nativo y, dentro del navegador se intenta loguear en el dashboard (<http://192.168.56.105/dashboard>).

Problema encontrado: Aparece un error que dice que "No tiene acceso para acceder al panel".

Sabiendo que las credenciales son correctas y la máquina está corriendo, el problema es que no estaba el servicio arrancado,

esto es, se había parado repentinamente debido a un problema de metadatos que a veces arroja Openstack en diferentes módulos.

- **Nova/Neutron caídos:** Se intenta crear una red nueva o, por el contrario, una instancia se crea y se queda en modo 'Scheduling' o 'Planificando'.

Problema encontrado: Parece que todo va correctamente, pero la ejecución se demora sustancialmente, al punto de tener que abortar el proceso. La conclusión es que Nova/Neutron se han caído y sus servicios se encuentran parados.

- **Solución general:** Reiniciar todos los servicios.

Esto puede hacerse en el momento en que se encuentra el fallo o por medio de editar el crontab (*crontab -e*) para que se haga cada 5 minutos una comprobación y se ejecute en caso de ser necesario. El script en cuestión consiste en listar los módulos o servicios de openstack y ver cuáles se encuentran corriendo, siendo posible verlo en el *Anexo 7.4*

- **Puertos huérfanos:** En algunos casos, después de eliminar dispositivos que inicialmente habían sido conectados, sus puertos no se borran adecuadamente y empezaba a existir ambigüedad en la arquitectura. Esto aparece en la *Figura 29*, cuyo contexto se basa en un entorno Openstack totalmente reinstalado y sin router, instancia o configuración:

```
root@serveropenstack:~# neutron port-list
neutron CLI is deprecated and will be removed in the future. Use openstack CLI instead.
```

id	name	tenant_id	mac_address	fixed_ips
3738f67a-0fd1-4517-a65e-2d8e5e6d43fe		d3aafe09ebec476aa5cc09dac06a7b58	fa:16:3e:c7:09:0f	{"subnet_id": "d4fc7302-30e1-4d8e-be55-aeb86103bdda", "ip_address": "10.0.0.1"}
7c222191-7b78-4e73-932c-7fd25bb0461f		d3aafe09ebec476aa5cc09dac06a7b58	fa:16:3e:4e:be:ab	{"subnet_id": "d4fc7302-30e1-4d8e-be55-aeb86103bdda", "ip_address": "10.0.0.2"}
				{"subnet_id": "dd3ac3f2-d5b4-487c-bf30-c9ff0dfe78db", "ip_address": "fd2a:f58:e6e:0:f816:3eff:fe4e:beab"}
81e6faa5-1b32-420a-a9b7-d58750143593		d3aafe09ebec476aa5cc09dac06a7b58	fa:16:3e:09:dc:62	{"subnet_id": "dd3ac3f2-d5b4-487c-bf30-c9ff0dfe78db", "ip_address": "fd2a:f58:e6e:0:1"}
c7fb4da8-4305-4699-a9d6-ef26e4d8a35e			fa:16:3e:6f:5f:ec	{"subnet_id": "f2d56241-42fb-4204-9f08-4e3b3dee8cbe", "ip_address": "172.24.4.158"}
				{"subnet_id": "562c74db-602c-4bd3-b904-bd87d1db4e26", "ip_address": "2001:db8::1"}

Figure 29: Puertos 'huérfanos'

- Estos dos problemas se suelen deber a la configuración *all_in_one*, en la cual no hay un servicio o nodo de metadatos directamente. Por lo tanto, es un mal menor en esta configuración.

3.2 Nagios - Monitorización del cluster

Nagios se encargará de gestionar los hosts de Openstack (el hipervisor, por tanto), esto es, las instancias que se crean en torno al servidor, que es donde se desplegará el Core [15] (implementación y servicio principal de Nagios). Dicho servidor será el Ubuntu virtualizado creado sobre el pc nativo (hardware), específicamente una instancia de OpenStack. Esto se ve en la *Figura 30*.

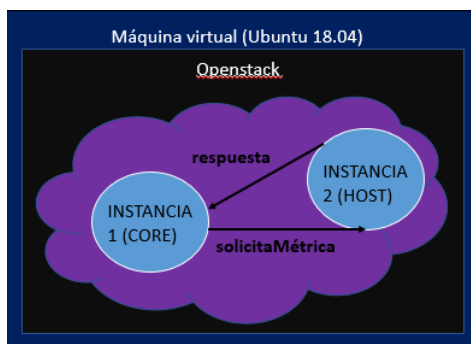


Figure 30: Relación de Nagios Core con la topología

En cuanto a las gestiones realizadas, van a ser una serie de ocho muestras de los servicios más básicos y de carácter general. En este caso:

***Carga del sistema - Usuarios activos - Servicio HTTP - PING
Partición root (espacio libre) - SSH - Swap - Procesos totales***

3.2.1 Implementación del core/server

Primero se lleva a cabo un script de instalación de los paquetes necesarios:

Example 16: Paquetes de Nagios

```
sudo apt update
sudo apt install autoconf gcc make unzip libgd-dev libmcrypt-dev \
libssl-dev dc snmp libnet-snmp-perl gettext
cd ~
curl -L -O \
https://github.com/NagiosEnterprises/nagioscore/archive/nagios-4.4.4.tar.gz
tar xzf nagios-4.4.4.tar.gz
cd nagioscore-nagios-4.4.4
./configure --with-httpd-conf=/etc/apache2/sites-enabled
```

Con esto, ya se tendría Nagios Core instalado. Ahora habría que compilar sus módulos y crear usuarios de uso de Nagios, así como permitir el acceso por medio del htpasswd (ya que se usa Apache):

Example 17: Compilación y configuración de Nagios

```
make all ; sudo make install--groups--users
sudo make install ; sudo make install--daemoninit
sudo make install--commandmode ; sudo make install--config
sudo make install--webconf ; sudo a2enmod rewrite
sudo a2enmod cgi ; sudo usermod -a -G nagios www-data
sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
sudo systemctl restart apache2
```

Ahora se instalan los plugins, que son los binarios que miden los datos:

Example 18: Descarga de plugins de Nagios

```
cd ~ ; curl -L -O \
https://nagios-plugins.org/download/nagios-plugins-2.2.1.tar.gz
tar xzf nagios-plugins-2.2.1.tar.gz ; cd nagios-plugins-2.2.1
./configure ; make ; sudo make install
```

Faltaría el plugin *check_nrpe* para que sea remota:

Example 19: Plugin *check_nrpe* de Nagios

```
cd ~ ; curl -L -O \
https://github.com/NagiosEnterprises/nrpe/releases/download/\
nrpe-3.2.1/nrpe-3.2.1.tar.gz
tar xzf nrpe-3.2.1.tar.gz ; cd nrpe-3.2.1 ; ./configure
make check_nrpe ; sudo make install--plugin
```

Ahora hay que definir los hosts remotos y el propio localhost (máquina nativa):

Example 20: Inicio de nagios

```
echo "cfg_dir=/usr/local/nagios/etc/servers" >> \
/usr/local/nagios/etc/nagios.cfg
sudo mkdir /usr/local/nagios/etc/servers
sudo nano /usr/local/nagios/etc/objects/contacts.cfg
```

En el fichero '*contacts.cfg*' se tiene que definir la notificación por mail (*Figura 31*):

```
define contact {
    contact_name      nagiosadmin      ;
    use               generic-contact  ;
    alias             Nagios Admin     ;
    email             cristiansainzdiego@gmail.com ;
}
```

Figure 31: Notificación por mail

Ahora falta añadir al catálogo de comandos el *check_nrpe* (Figura 32):

```
define command{
    command_name check_nrpe
    command_line $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
}
```

Figure 32: Fichero /usr/local/nagios/etc/objects/commands.cfg

Finalmente se define el fichero del localhost (/usr/local/nagios/etc/objects/localhost.cfg), con sus datos (Figura 33) y servicios a monitorizar (Figura 34). Después se hace 'systemctl start nagios'.

```
define host {
    use linux-server

    host_name localhost
    alias localhost
    address 127.0.0.1
}

#####
#
# HOST GROUP DEFINITION
#
#####
# Define an optional hostgroup for Linux machines
define hostgroup {
    hostgroup_name linux-servers
    alias Linux Servers
    members localhost
}
```

Figure 33: Definición de información de localhost

```
define service {
    use local-service ; T
    host_name localhost
    service_description PING
    check_command check_ping!100.0,20%!500.0
}

# Define a service to check the disk space of the root
# on the local machine. Warning if < 20% free, critical
# < 10% free space on partition.
define service {
    use local-service ; T
    host_name localhost
    service_description Root Partition
    check_command check_local_disk!20%!10%!/
}
```

Figure 34: Ejemplo de servicios del localhost

3.2.2 Generación de agentes para hosts remotos

En primer lugar, se crea un usuario para poder interaccionar con Nagios. A su vez, se descarga el plugin nrpe dentro del host (máquina remota o instancia cirros/ubuntu creada) con la intención de poder permitir la monitorización y ver si se encuentra disponible el host (ya que este plugin sirve para conectar el host con el server):

Example 21: Instalacion en host remoto

```
sudo useradd nagios
<instalacion de plugins ya vista>
cd ~
curl -L -O \
https://github.com/NagiosEnterprises/nrpe/releases/download/ \
nrpe-3.2.1/nrpe-3.2.1.tar.gz
tar xzf nrpe-3.2.1.tar.gz
cd nrpe-3.2.1
./configure
make nrpe
sudo make install--daemon
sudo make install--config
sudo make install--init
```

Ahora debería configurarse el fichero del nrpe (`/usr/local/nagios/etc/nrpe.cfg`), hecho que se puede ver en las *Figuras 35/36/37*, en las cuales se definen los comandos implícitos al nrpe para monitorizar (*Figura 37*), así como el host servidor de Nagios para permitir su acceso (*Figura 35/36*):

```
allowed_hosts=127.0.0.1,:::1,192.168.56.105
```

Figure 35: Hosts permitidos para monitorizar

```
server_address=0.0.0.0
```

Figure 36: Server address

```
command[check_users]=/usr/local/nagios/libexec/check_users -w 5 -c 10
command[check_load]=/usr/local/nagios/libexec/check_load -r -w .15,.10,.05 -c .30,.25,.20
command[check_vdal]=/usr/local/nagios/libexec/check_disk -w 20% -c 10% -p /dev/vdal
command[check_zombie_procs]=/usr/local/nagios/libexec/check_procs -w 5 -c 10 -s 2
command[check_total_procs]=/usr/local/nagios/libexec/check_procs -w 150 -c 200
```

Figure 37: Comandos del check_nrpe

De este modo, el host remoto ya queda configurado (a faltar de hacer un `systemctl restart check_nrpe`), ya que tendría el `check_nrpe`, los plugins para monitorizar y la configuración necesaria para permitir que el Nagios Core (192.168.56.105) lo monitorice.

Solamente faltaría irse a la máquina host de Openstack, aquella que es el core de Nagios y, en el directorio de 'servers' al que se hizo un `mkdir`, copiar el fichero de localhost (aquel con la configuración y servicios), cambiando solamente la información IP y del grupo. De esta forma, se tendría cada host monitorizando los mismos servicios que localhost, pero cambiando el target, tal y como puede verse en la *Figura 38*:

```
Example 22: host remoto
cp /usr/local/nagios/etc/objects/localhost.cfg \
/usr/local/nagios/etc/servers/ubuntu_host.cfg

define host {
    use                linux-server
    host_name          ubuntu-host
    alias              ubuntu-host
    address            172.24.4.179
}
```

Figure 38: Cambios en la información del fichero

Finalmente, se hace un `systemctl restart nagios`.

Nota: Es necesario abrir el puerto 5666 bidireccionalmente para permitir la monitorización. Se puede ver en la *Figura 39* que no se podía conectar. A su vez, en la *Figura 40* se puede ver que el proceso está activo y en la 41 y 42 se ve de qué manera se soluciona:

```
root@devstack:~/nrpe-3.2.1# /usr/local/nagios/libexec/check_nrpe -H 192.168.233.130
CHECK_NRPE STATE CRITICAL: Socket timeout after 10 seconds.
```

Figure 39: Fallo al conectar con el host

```
root@ubuntu-host:~/nrpe-3.2.1# ps ax | grep nrpe
13834 ?        Ss      0:00 /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -f
13880 pts/0    S+      0:00 grep --color=auto nrpe
```

Figure 40: Proceso activo

```
root@devstack:~/nrpe-3.2.1# firewall-cmd --zone=public --add-port=5666/tcp
success
root@devstack:~/nrpe-3.2.1# firewall-cmd --zone=public --add-port=5666/tcp --permanent
success
root@devstack:~/nrpe-3.2.1#
```

Figure 41: Solución de fallo de conexión

```
root@devstack:~/nrpe-3.2.1# iptables -I INPUT -p tcp --destination-port 5666 -j ACCEPT
```

Figure 42: Solución (II) de fallo de conexión

3.2.3 Demostración de funcionamiento

Este apartado consistirá en una serie de capturas del funcionamiento de Nagios, que ya quedó configurado anteriormente, con la intención de que se vea (y se explique) un poco su funcionalidad o posibilidades por parte del administrador.

En primer lugar, se tiene una figura (*Figura 43*) en la que se ven cómo los hosts están activos. Si se profundiza, se aprecian 3 hosts, dos de los cuales aparecen como disponibles (*localhost*, *ubuntu-host*), además de un tercero que aparece en rojo por no ser accesible. Esta figura es una prueba para intentar monitorizar hosts, ya que el tercero se dota de una *IP* inexistente en OpenStack mientras que esto no ocurre para las otras instancias. Por lo tanto, se ve que Nagios accede a OpenStack correctamente, estando bien integrado y encontrando adecuadamente los hosts creados.

Limit Results:

Host ♦♦	Status ♦♦	Last Check ♦♦	Duration ♦♦	Status Information
localhost	UP	03-29-2021 02:53:59	0d 4h 6m 35s	PING OK - Packet loss = 0%, RTA = 0.13 ms
ubuntu-host	UP	03-29-2021 02:52:25	0d 0h 9m 47s	PING OK - Packet loss = 0%, RTA = 1.06 ms
ubuntu-host_inventado	DOWN	03-29-2021 02:56:57	0d 0h 1m 19s	CRITICAL - Host Unreachable (172.24.4.180)

Figure 43: Hosts activos

Por otra parte, se puede ver que es posible acceder a los servicios de los hosts disponibles en la *Figura 44*, en la cual aparecen para *localhost* y *ubuntu-host* todos los servicios monitorizados. Cabe destacar que pueden aparecer en estado óptimo (ok), con Warning (advertencia) y en modo crítico (algo va mal, por ejemplo si la memoria disponible es baja). Véase que aparece información más extensa a la derecha de los propios servicios, haciendo posible que con una frase breve se localice el error o se entienda el estado actual.

Host ♦♦	Service ♦♦	Status ♦♦	Last Check ♦♦	Duration ♦♦	Attempt ♦♦	Status Information
localhost	Current Load	OK	06-28-2021 16:19:52	28d 22h 46m 27s	1/4	OK - load average: 0.00, 0.02, 0.06
	Current Users	OK	06-28-2021 16:20:30	28d 22h 45m 49s	1/4	USERS OK - 1 users currently logged in
	HTTP	WARNING	06-28-2021 16:19:37	28d 2h 8m 52s	4/4	HTTP WARNING: HTTP/1.1 403 Forbidden - 460 bytes in 0.001 second response time
	PING	OK	06-28-2021 16:21:07	28d 22h 44m 34s	1/4	PING OK - Packet loss = 0%, RTA = 0.06 ms
	Root Partition	OK	06-28-2021 16:16:45	28d 22h 43m 57s	1/4	DISK OK - free space: / 6251 MB (66.73% inode=76%):
	SSH	OK	06-28-2021 16:17:22	28d 22h 43m 19s	1/4	SSH OK - OpenSSH_7.9p1 Debian-10+deb10u2 (protocol 2.0)
	Swap Usage	CRITICAL	06-28-2021 16:18:00	28d 23h 2m 42s	4/4	SWAP CRITICAL - 0% free (0 MB out of 0 MB) - Swap is either disabled, not present, or of zero size.
	Total Processes	OK	06-28-2021 16:18:37	28d 22h 47m 4s	1/4	PROCS OK: 34 processes with STATE = RSZDT
	Current Load	OK	06-28-2021 16:19:15	0d 0h 7m 36s	1/4	OK - load average: 0.00, 0.02, 0.06
	Current Users	OK	06-28-2021 16:20:11	0d 0h 11m 40s	1/4	USERS OK - 1 users currently logged in
ubuntu-host	HTTP	CRITICAL	06-28-2021 16:18:48	0d 0h 8m 3s	4/4	connect to address 172.24.4.179 and port 80: Connection refused
	PING	OK	06-28-2021 16:21:26	0d 0h 10m 25s	1/4	PING OK - Packet loss = 0%, RTA = 0.69 ms
	Root Partition	OK	06-28-2021 16:17:03	0d 0h 9m 48s	1/4	DISK OK - free space: / 6251 MB (66.72% inode=76%):
	SSH	OK	06-28-2021 16:17:41	0d 0h 9m 10s	1/4	SSH OK - OpenSSH_7.9p1 Debian-10+deb10u2 (protocol 2.0)
	Swap Usage	CRITICAL	06-28-2021 16:21:18	0d 0h 5m 33s	4/4	SWAP CRITICAL - 0% free (0 MB out of 0 MB) - Swap is either disabled, not present, or of zero size.
	Total Processes	OK	06-28-2021 16:18:56	0d 0h 7m 55s	1/4	PROCS OK: 34 processes with STATE = RSZDT

Figure 44: Servicios configurados para los hosts

También es posible recoger estadísticas del uso y disponibilidad de las instancias. En este caso, se puede ver en la *Figura 45* como ha evolucionado la disponibilidad de la instancia central (*localhost*). Ciertamente no se nota variabilidad, ya que ha estado prácticamente siempre disponible desde que se creó. Solamente se han experimentado momentos de pérdida de conexión a la hora de configurar los ficheros de Nagios, ya que en algún momento se ha intentado hacer un *restart* de su servicio y algún fallo de sintaxis abortaba dicho servicio. Es por ello que dicho estado solamente se mantuvo durante unos 20 minutos a lo largo del tiempo.

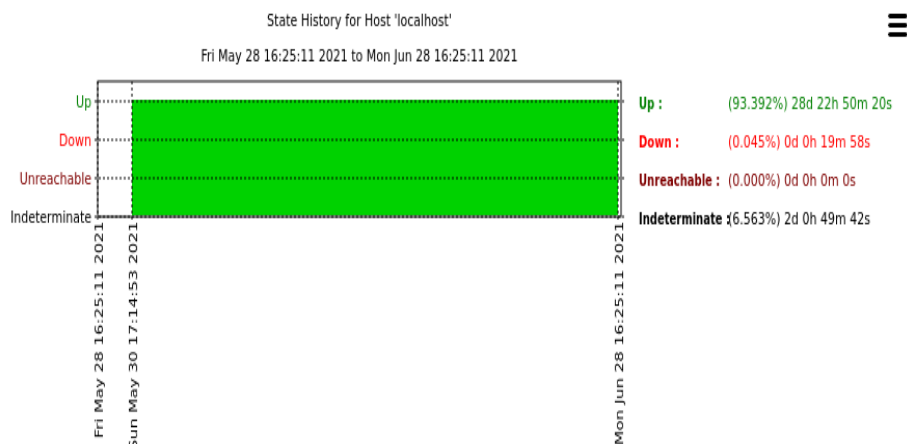


Figure 45: Reports con gráfica de disponibilidad

A continuación, puede verse en la *Figura 46* de qué manera ha quedado la topología monitorizada, (siendo en este caso bastante sencilla ya que solamente se monitoriza como ejemplo la instancia *ubuntu-host*).



Figure 46: Mapa de Nagios

Además, en caso de tener que debuggear un posible fallo, aparece un log con los eventos más importantes de todas las máquinas, tal y como puede verse en la *Figura 47*.

July 08, 2021 11:00

[07-08-2021 11:48:44] Nagios 4.4.6 starting... (PID=550)

June 28, 2021 16:00

 [06-28-2021 16:26:24] Caught SIGTERM, shutting down...
 [06-28-2021 16:16:18] SERVICE ALERT: ubuntu-host:Swap Usage:CRITICAL:HARD:4:SWAP CRITICAL - 0% free (0 MB out of 0 MB) - Swap is either disabled, not present, or of zero size.
 [06-28-2021 16:15:18] SERVICE ALERT: ubuntu-host:Swap Usage:CRITICAL:SOFT:3:SWAP CRITICAL - 0% free (0 MB out of 0 MB) - Swap is either disabled, not present, or of zero size.
 [06-28-2021 16:14:18] SERVICE ALERT: ubuntu-host:Swap Usage:CRITICAL:SOFT:2:SWAP CRITICAL - 0% free (0 MB out of 0 MB) - Swap is either disabled, not present, or of zero size.

Figure 47: Log de eventos de Nagios

Finalmente, otro apartado esencial de la monitorización de Nagios es el uso de un planificador interno, que es aquel que encola los checkings o comprobaciones de los servicios configurados para ser monitorizados en cada uno de los hosts. Como puede verse en la *Figura 48*, aparece una entrada por cada fila, aportando información esencial como es el caso del host de destino, el servicio en cuestión a comprobar, y un timestamp con la última (y próxima) comprobación. Si bien esto es automático, es posible abortar comprobaciones o incluso programarlas por uno mismo para una hora específica.

Entries sorted by **next check time** (ascending)

Host ★★	Service ★★	Last Check ★★	Next Check ★★	Type	Active Checks	Actions
localhost	Total Processes	06-28-2021 16:18:37	06-28-2021 16:23:37	Normal	ENABLED	✗ ⌚
ubuntu-host	HTTP	06-28-2021 16:18:48	06-28-2021 16:23:48	Normal	ENABLED	✗ ⌚
ubuntu-host	Total Processes	06-28-2021 16:18:56	06-28-2021 16:23:56	Normal	ENABLED	✗ ⌚
localhost		06-28-2021 16:19:15	06-28-2021 16:24:15	Normal	ENABLED	✗ ⌚
ubuntu-host	Current Load	06-28-2021 16:19:15	06-28-2021 16:24:15	Normal	ENABLED	✗ ⌚
localhost	HTTP	06-28-2021 16:19:37	06-28-2021 16:24:37	Normal	ENABLED	✗ ⌚
localhost	Current Load	06-28-2021 16:19:52	06-28-2021 16:24:52	Normal	ENABLED	✗ ⌚
ubuntu-host	Current Users	06-28-2021 16:20:11	06-28-2021 16:25:11	Normal	ENABLED	✗ ⌚
localhost	Current Users	06-28-2021 16:20:30	06-28-2021 16:25:30	Normal	ENABLED	✗ ⌚
localhost	PING	06-28-2021 16:21:07	06-28-2021 16:26:07	Normal	ENABLED	✗ ⌚
ubuntu-host	Swap Usage	06-28-2021 16:21:18	06-28-2021 16:26:18	Normal	ENABLED	✗ ⌚
ubuntu-host	PING	06-28-2021 16:21:26	06-28-2021 16:26:26	Normal	ENABLED	✗ ⌚
localhost	Root Partition	06-28-2021 16:21:45	06-28-2021 16:26:45	Normal	ENABLED	✗ ⌚
ubuntu-host	Root Partition	06-28-2021 16:22:03	06-28-2021 16:27:03	Normal	ENABLED	✗ ⌚
localhost	SSH	06-28-2021 16:22:22	06-28-2021 16:27:22	Normal	ENABLED	✗ ⌚
ubuntu-host	SSH	06-28-2021 16:22:41	06-28-2021 16:27:41	Normal	ENABLED	✗ ⌚
localhost	Swap Usage	06-28-2021 16:23:00	06-28-2021 16:28:00	Normal	ENABLED	✗ ⌚
ubuntu-host		06-28-2021 16:23:18	06-28-2021 16:28:18	Normal	ENABLED	✗ ⌚

Figure 48: Planificador de monitorización

4 Conclusión

Finalmente, se ha conseguido llegar a desplegar la red simulada de un entorno empresarial por medio de un servidor (la máquina virtual con el entorno OpenStack), entendiendo el servidor como una de las máquinas de la empresa con, a poder ser, una buena capacidad de recursos para poder operar a la perfección. Todo esto se ha llevado a cabo de una manera segura, permitiendo solamente una serie de puertos en el tráfico, siendo esto necesario para llevar a cabo la monitorización de las máquinas (así como llevar planificaciones de tareas y checks de servicios) que se ha realizado posteriormente con Nagios, así como permitiendo únicamente una IP de acceso (aquella del VirtualHost) a la red creada.

Una última característica conseguida en el entorno final es la persistencia en la configuración de la red, esto es, la flexibilidad / adaptabilidad del sistema, ya que al emplear y configurar las interfaces de red de la manera vista en la configuración, se puede exportar el entorno y llevarse a otra máquina que actúe de servidora (para poder tener backups y no tener que configurar todo de nuevo), hecho que al usar interfaces más fáciles de configurar (como la bridge only) no era posible y añadía mucho overhead de tiempos y esfuerzo a la hora de llevar a cabo replicación de instancias. Esto es algo así como el directorio activo LDAP, pero con configuraciones de red en vez de datos de usuarios y reglas.

5 Futuras líneas de trabajo

El proyecto, tal y como se explicará en breves, tiene un cuello de botella en la memoria RAM, lo cual lleva a una serie de alternativas, que podrían ser implementaciones finales en un futuro.

En primer lugar, ya se ha podido ver en el apartado de configuración que el entorno desplegado es completo, total. Al decir esto, se hace referencia a que la implementación es *all in one*, lo cual significa que todos los módulos de OpenStack se implementan en una misma máquina host.

La idea es que cuanto más abstracto todo mejor, ya que esto permite más flexibilidad a la hora de configurar cada uno de los módulos por separado, lo cual lleva a la primera línea de trabajo posible. De este modo, si bien sigue funcionando como un conjunto ya que es transparente de cara al usuario final (como todos los entornos distribuidos), se podrían crear una serie de máquinas virtuales, cada una operable por sí misma, de modo que no todos los módulos se encuentran en una máquina. *Para más información acerca de una posible alternativa empleando este método, vaya al Anexo 7.2.*^[16]

En segundo lugar, se podría adoptar una línea de trabajo en la cual se use docker/kubernetes, siendo ambas alternativas para la configuración y despliegue del Cloud.

En cuanto a docker, es una alternativa para mejorar notablemente el uso de recursos del sistema desplegado. Considerando que el cuello de botella de este proyecto es la memoria RAM, una alternativa para haberlo solucionado sería desplegar la máquina virtual de Mano como un contenedor. Un contenedor en docker es un proceso del Sistema Operativo que encapsula un funcionamiento (en este caso, encapsularía la ejecución de Mano). La gracia de docker es que estos contenedores se pueden poner a correr y posteriormente eliminarse, tan simple como tratarlos como a procesos normales. *Para más información acerca de una posible alternativa empleando este método, vaya al Anexo 7.3.*^[17]

La otra alternativa para mejorar la implementación sería hacer uso de Kubernetes, que sirve para automatizar este despliegue en entornos normalmente mucho más grandes. Este software mejora la escalabilidad vertical del sistema desplegado, pero requiere de muchos meses para poder emplearse con cierta solvencia. Consecuentemente, si bien constituye una alternativa, no sería muy recomendable para gente que está aprendiendo.

Finalmente, cabe destacar que habría sido beneficioso implementar el software Mano, que es un Sistema NFV (Network Function Virtualizer), sirviendo este para poder definir las redes de OpenStack de manera automática. Dicho esto, al igual que se ha tenido que configurar todo manualmente, con Mano (una vez estando OpenStack desplegado) se hubiera podido descargar plantillas o

configurar jerarquías que más tarde se crean y despliegan automáticamente en OpenStack. Esto constaba de una serie de comandos para poder conectar la IP de la máquina que tenía OpenStack con otra máquina virtual nueva (siendo los comandos ejecutados en la máquina de MANO). En cualquier caso, para hacer esto se debían tener en paralelo las máquinas de Mano y OpenStack en un mismo momento, lo cual lleva a la problemática por la cual no se ha probado a hacer esto una vez configurado el entorno manualmente, que es el problema de RAM ya mencionado.

Requerimientos de memoria (RAM)	Máquina virtual
4GB mínimo	OpenStack
4GB mínimo	Mano

Esta información lleva a la siguiente conclusión:

Memoria RAM total en paralelo
8GB mínimo
Memoria disponible en el HardWare
8GB de manera nativa
Memoria restante
0GB con ejecución simultánea

Como es de esperar, si no se le deja a Windows (SO nativo) nada de memoria, no puede operar. Por lo tanto, la convergencia en estos casos dicta que libera memoria de los procesos que la están empleando para que el Sistema base funcione adecuadamente. En este caso, lo que ocurre es que la máquina de mano la cierra, abortando el proceso que la sostiene. *Para más información acerca de una posible alternativa empleando este método, vaya al Anexo 7.1.*[18]

6 Bibliografía

References

- [1] *¿Qué es AWS?* (s. f.). Amazon Web Services, Inc. <https://aws.amazon.com/es/what-is-aws/>
- [2] *Cloud Computing Services — Google Cloud.* (s. f.). Google Cloud. <https://cloud.google.com/>
- [3] *What is Microsoft Azure Cloud & What is It Used for?* (s. f.). Datamation. <https://www.datamation.com/cloud/microsoft-azure-cloud/>
- [4] *What is OpenStack?* (s. f.). Opensource.com. <https://opensource.com/resources/what-is-openstack>
- [5] *Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution.* (s. f.). Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution. <https://www.zabbix.com/>
- [6] *Quiénes somos — Pandora FMS.* (s. f.). Pandora FMS — The flexible monitoring software. <https://pandorafms.com/es/quienes-somos/>
- [7] *Linux Monitoring Software and Tools - Nagios.* (s. f.). Nagios. <https://www.nagios.com/solutions/linux-monitoring/>
- [8] *El concepto de OpenStack.* (s.f.). Red Hat - We make open source technologies for the enterprise. <https://www.redhat.com/es/topics/openstack>
- [9] *Nagios - Network, Server and Log Monitoring Software.* (s.f.). Nagios. <https://www.nagios.com/>
- [10] *VirtualBox (configurando la red) - El Pinguino y la Taza.* (s.f.). El Pinguino y la Taza. <https://www.pinguytaz.net/index.php/2016/11/20/virtualbox-configurando-la-red/>
- [11] *Networking in too much detail —RDO.* (s.f.). RDO. <https://www.rdoproject.org/networking/networking-in-too-much-detail/>
- [12] *Chapter 9. Connect an instance to the physical network Red Hat OpenStack Platform 10 — Red Hat Customer Portal.* (s.f.). Red Hat Customer Portal. https://access.redhat.com/documentation/en-us/red_hat_openshift_platform/10/html/networking_guide/sec-connect-instance
- [13] *Openstack pike linuxbridge replaced with openvswitch - Programmer Sought.* (s.f.). Programmer Sought. <https://www.programmersought.com/article/1824577065/>

- [14] *OpenStack Docs: Command List.* (s.f.). OpenStack Docs: Wallaby. <https://docs.openstack.org/python-openstackclient/pike/cli/command-list.html>
- [15] *Tutorial - Nagios Installation on Ubuntu Linux — Step by Step.* (s. f.). TechExpert. <https://techexpert.tips/nagios/nagios-installation-on-ubuntu-linux/>
- [16] *Multi-node OpenStack RDO on RHEL and Hyper-V - Part 1 - Cloudbase Solutions.* (s.f.). Cloudbase Solutions. <https://cloudbase.it/rdo-multi-node/>
- [17] *Tutorial de Docker: instalar y gestionar la plataforma de contenedores.* (s.f.). IONOS Digitalguide. <https://www.ionos.es/digitalguide/servidores/configuracion/tutorial-docker-instalacion-y-primeros-pasos/>
- [18] *3. Installing OSM — Open Source MANO 6.0 documentation.* (s.f.). OSM. <https://osm.etsi.org/docs/user-guide/03-installing-osm.html>

7 Anexo

En este apartado se van a aclarar, con un mínimo más de profundidad, de qué manera funcionarían las líneas de proyectos futuros, así como otras aclaraciones.

7.1 Mano (OSM)

En primer lugar, la distribución de la arquitectura sería tal y como puede verse en la *Figura 49*:

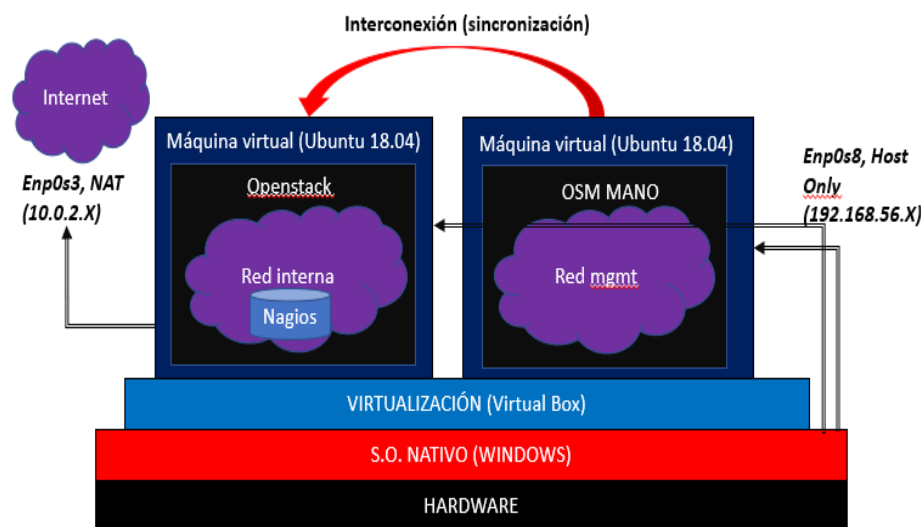


Figure 49: Arquitectura para Mano

La idea se basa en implementar una nueva máquina virtual (dentro de Virtual Box, no dentro de la máquina con Openstack), con las mismas interfaces de red ya explicadas. Luego se tiene que instalar:

Example 23: Script de instalación de Mano

```
wget https://osm-download.etsi.org/ftp/osm-10.0-ten/install_osm.sh
chmod +x install_osm.sh
./install_osm.sh 2>&1 | tee osm_install_log.txt
```

La idea ahora es construir una red (*Figura 50*) y subnet implícita (*Figura 51*) que permitan el soporte de Mano para la conexión con Openstack, lo cual se puede hacer con estos dos comandos de las Figuras mencionadas anteriormente:

```

root@devstack:~# neutron net-create mgmt --provider:network_type=vlan --shared
neutron CLI is deprecated and will be removed in the future. Use openstack CLI instead.
Created a new network:
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | True |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2021-03-26T00:33:49Z |
| description | |
| id | 66755179-ffc9-4a6a-8413-831f44d076d4 |
| ipv4_address_scope | |
| ipv6_address_scope | |
| is_default | False |
| mtu | 1500 |
| name | mgmt |
| port_security_enabled | True |
| project_id | e3dd25ba0f844acf8144ea483d907caf |
| provider:network_type | vlan |
| provider:physical_network | public |
| provider:segmentation_id | 1 |
| revision_number | 1 |
| router:external | False |
| shared | True |
| status | ACTIVE |
| subnets | |
| tags | |
| tenant_id | e3dd25ba0f844acf8144ea483d907caf |
| updated_at | 2021-03-26T00:33:49Z |
+-----+-----+

```

Figure 50: Definición de red troncal

```

root@devstack:~# neutron subnet-create --name subnet-mgmt mgmt 10.208.0.0/24 --allocation-pool start=10.208.0.2,end=10.208.0.254
neutron CLI is deprecated and will be removed in the future. Use openstack CLI instead.
Created a new subnet:
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | {"start": "10.208.0.2", "end": "10.208.0.254"} |
| cidr | 10.208.0.0/24 |
| created_at | 2021-03-26T00:34:34Z |
| description | |
| dns_nameservers | |
| enable_dhcp | True |
| gateway_ip | 10.208.0.1 |
| host_routes | |
| id | 45490f77-133b-4e67-9ff7-c38341b13f11 |
| ip_version | 4 |
| ipv6_address_mode | |
| ipv6_ra_mode | |
| name | subnet-mgmt |
| network_id | 66755179-ffc9-4a6a-8413-831f44d076d4 |
| project_id | e3dd25ba0f844acf8144ea483d907caf |
| revision_number | 0 |
| service_types | |
| subnetpool_id | |
| tags | |
| tenant_id | e3dd25ba0f844acf8144ea483d907caf |
| updated_at | 2021-03-26T00:34:34Z |
+-----+-----+

```

Figure 51: Definición de subnet

Finalmente solo falta integrarlo (*Figura 52*) con ambas máquinas conectadas, que es el problema existente, si bien los pasos son estos mismos:

```

ot@manoserver:~$ osm vim-create --name stack --user admin --password sainzCRIS --auth_url http://192.168.56.105/identity --tenant admin --account_type openstack --confi
g='(security_groups: ubuntu personalizado, keypair: llave)'
19445abc-847c-4b5f-9354-c95da2947be9
root@manoserver:~$

```

Figure 52: Integración de Mano con Openstack

Como puede verse, hay varios parámetros:

- **password:** contraseña de openstack
- **auth_url 192.168.56.105/identity:** URL de acceso a la API (disponible en Openstack: *192.168.56.105/dashboard*).
- **tenant admin:** proyecto admin.
- **account_type openstack:** tipo de conexión o integración.
- config: configuración extra. Parámetros:
 - **security_groups: ubuntu_personalizado.**

Es el grupo empleado en la integración, creado en el apartado de desarrollo.

- **keypair: llave.**

Llave para tener acceso a las instancias.

Una vez hecho esto, ya solamente falta ir a la página de OSM Mano, descargar los paquetes de VNFs (Virtual Network Functions, es decir, la estructura a implementar en Openstack) y NS (Network Service, es decir, las funciones de red). Al descargarse, se va al apartado de Mano de Importar VNFs, y se importan ambos schemas. Cuando esté finalizado, se habrá importado dicha estructura en Openstack.

Recuérdese que para que Mano funcione, Openstack debe estar funcionando. En el caso de este proyecto, la configuración e integración de Mano es correcta, y Openstack está funcionando adecuadamente, si bien el problema es que las máquinas se abortan al estar en paralelo por la RAM necesaria. En cualquier caso, estos comandos son funcionales.

7.2 Entorno distribuido (Modularizado)

Una posible distribución de los módulos de Openstack sería:

- **Nodo 1 (Controlador):**
 - Módulo Nova (servicios).
 - Módulo Keystone.
 - Módulo Horizon.

- Módulo Glance.
- Módulo Cinder.
- **Nodo 2 (Red):**
 - Módulo Neutron.
- **Nodo 3 (Cómputo):**
 - Módulo Nova (Compute)

Lo cual haría que la estructura estuviera de la siguiente manera (*Figura 53*), siendo cada nodo una máquina virtual o instancia:

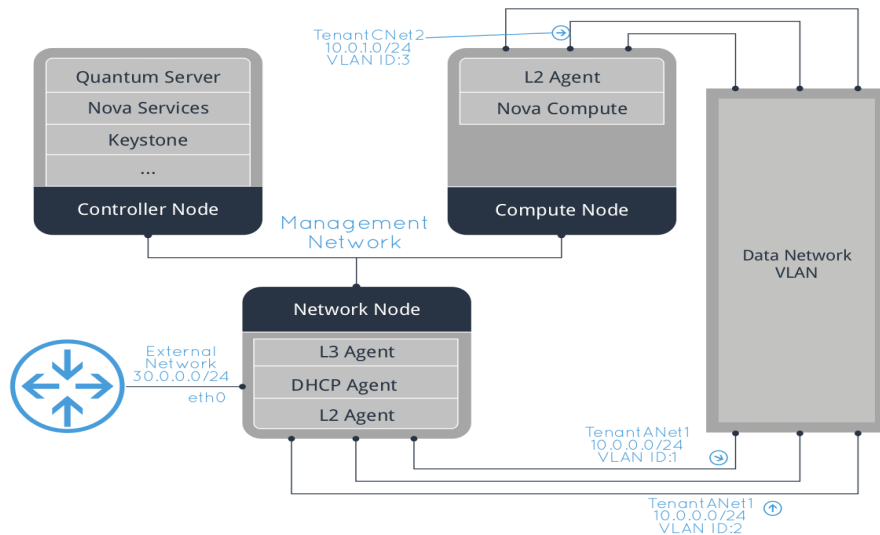


Figure 53: Arquitectura multi-nodo con RDO

Incluso sería posible poner un nodo para el control de metadatos, con los servicios *metad* ya conocidos, aunque esto es un poco más avanzado aún.

En cualquier caso, si bien tiene muchos beneficios este despliegue, es muy costoso y la configuración (ya que por defecto es *all_in_one*) se vuelve muy tediosa, teniendo que ser el propio administrador el que lleve a cabo todas las conexiones entre módulos (con lo que ello implica: puertos, seguridad, IP's aceptadas, forwarding, entre otros).

7.3 Entorno con kubernetes/docker

Para poder implementar mano se utiliza el formato Dockerfile, que es como un Makefile pero con directivas para crear el proceso. Dicho de otro modo, cuando se lanza el archivo dockerfile, todo lo que hay dentro se ejecuta como si de un

script se tratara, guardando en un proceso su resultado. Como ejemplo se tiene un fichero *Dockerfile* sencillo, que permite actualizar el sistema y hacer varios ejecutables por medio del *chmod +x* (*Figura 54*):

```
FROM node:4-slim

#
# Inicio automático del servidor Node.js con el contenedor
#
ENTRYPOINT ["/usr/local/bin/wlpn-server", "run", "defaultServer"]

#
# Añadir colectivos que habilitan scripts
#
ADD joinMember /opt/ibm/docker/
RUN chmod +x /opt/ibm/docker/joinMember
ADD removeMember /opt/ibm/docker/
RUN chmod +x /opt/ibm/docker/removeMember

#
# Actualizar el OS y JRE
#
RUN apt-get update

#
# Instalar miembro de colectivo APIConnect
#
RUN npm install -g apiconnect-collective-member
```

Figure 54: Ejemplo de dockerfile

Como puede verse, el formato es muy sencillo. Ahora bien, no es interactivo, lo cual significa que todo se lleva a cabo por comandos directos, sin poder escribir nada cuando se aplica el Dockerfile. Por lo tanto, al hacer una instalación, por ejemplo, se debe poner el parámetro *-y*. Finalmente, la etiqueta *FROM* sirve para coger la imagen base, siendo posible poner ubuntu-18.04 y te cogería el SO dado de los repositorios de docker. Por tanto, ese sería el primer paso para crear el proceso conteniendo a Mano. Posteriormente, habría que intentar instalarlo desde cero y hacer esos comandos en un script contenido en el fichero Dockerfile.

7.4 Script de servicios Openstack

Tal y como puede verse en la *Figura 55*, así queda la monitorización de servicios con el script sencillo del que ya se habló:

```
root@devstack:~# for i in $(systemctl list-unit-files --type=service | grep devstack | cut -d " " -f 1); do systemctl status $i; done
● devstack@c-api.service - Devstack devstack@c-api.service
   Loaded: loaded (/etc/systemd/system/devstack@c-api.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2021-03-25 11:27:32 CET; 8min ago
     Main PID: 3034 (uwsgi)
    Status: "uwsgi is ready"
       Tasks: 5 (limit: 4660)
    CGroup: /system.slice/system-devstack.slice/devstack@c-api.service
            └─3034 cinder-apiuwsgi master
              └─3046 cinder-apiuwsgi worker 1
                └─3047 cinder-apiuwsgi worker 2
```

Figure 55: Servicios de Openstack activos